Politechnika Wrocławska

Wydział Elektroniki, Fotoniki i Mikrosystemów

Projektowanie Algorytmów i Metody Sztucznej Inteligencji

Kółko i krzyżyk

Projekt 3 - zadanie na ocenę bdb

Prowadzący:
Dr inż. Łukasz Jeleń

Wykonała:
Zuzanna Mejer, 259382

Wrocław, 9 czerwca 2022r.

Spis treści

1. Wprowadzenie

Zadanie polegało na zaimplementowaniu gry Kółko i Krzyżyk z wykorzystaniem technik sztucznej inteligencji - algorytmu Minimax z alfa-beta cięciami. Użytkownik miał mieć możliwości definiowania rozmiaru planszy wraz z ilością znaków w rzędzie.

2. Opis tworzonej gry

Kółko i Krzyżyk to gra o sumie zerowej, w której bierze udział dwóch użytkowników. W tym przypadku jest to gracz i komputer. Obydwoje gracze dążą do ustawienia swoich znaków w rzędzie, kolumnie lub po przekątnej tak, aby zachować ciągłość swoich znaków (żeby znak przeciwnika nie przeszkodził w tworzeniu linii). Jednocześnie, gracze dążą do tego, żeby przerwać ciągłość znaków przeciwnika w linii. W celu stworzenia gry w Kółko i Krzyżyk przyjęto parę założeń:

- Grę zawsze rozpoczyna Kółko.
- Użytkownik nie może wybrać czy jest Kółkiem, czy Krzyżykiem. Użytkownik zawsze jest Kółkiem i zatem zawsze rozpoczyna grę.
- Przed rozpoczęciem gry użytkownik musi zdefiniować rozmiar kwadratowej planszy, to znaczy ile chce mieć kratek w rzędzie.
- W zależności od wybranego rozmiaru planszy, do wygranej należy ułożyć w rzędzie, kolumnie lub po przekątnej tyle znaków, ile wynosi liczba kratek w jednej linii na planszy. To znaczy, że trzeba ze znaków ułożyć ciągłą linię od jednego końca planszy, do drugiego.

3. Graficzny interfejs użytkownika

W celu ułatwienia użytkownikowi obsługi gry, stworzono graficzny interfejs. Wykorzystano w tym celu bibliotekę programistyczną *SFML* (Simple and Fast Multimedia Library).

3.1. Kółko i Krzyżyk

Kółko i Krzyżyk zostały zaimplementowane i rysowane jako litery „X” i „O” czcionką Arial o odpowiednio dobranym rozmiarze tak, żeby wpasowywały się w środek kratki, niezależnie od liczby kratek na planszy. Przedstawia to poniższy kod:

```
1     sf::Font font;
2     if (!font.loadFromFile("arial.ttf"))
3     {
4         std::cout << "Nie mamy takiej czcionki\n";
5     }
6
7     sf::Text circle;
8     circle.setFont(font);
9     circle.setString("O");
10    circle.setCharacterSize(one_cell_size);
11    circle.setFillColor(sf::Color::Black);
```

```
12      circle.setLineSpacing(0);
13      circle.setLetterSpacing(0);
14
15
16      sf::Text cross;
17      cross.setFont(font);
18      cross.setString("X");
19      cross.setCharacterSize(one_cell_size);
20      cross.setFillColor(sf::Color::Black);
21      cross.setLineSpacing(0);
22      cross.setLetterSpacing(0);
```

3.2. Rysowanie planszy

Rysowanie planszy odbywa się tylko raz na początku gry. Po podaniu przez użytkownika rozmiaru planszy, program dostosowuje rozmiar jednej kratki do wielkości okna, zdefiniowanej jako zmienna globalna: *constexpr int one_side_number_of_cells = 800;*. Następnie rysuje kratki za pomocą prostokątów o odpowiednim rozmiarze i odpowiedniej rotacji. Jest to przedstawione poniżej:

```
1  void draw_board(sf::RenderWindow &window, int cells)
2  {
3      int one_cell_size = window.getSize().x / cells;
4      int tmp = one_cell_size;
5      sf::RectangleShape rectangle(sf::Vector2f(5, one_side_number_of_cells));
6      rectangle.setFillColor(sf::Color::Black);
7
8      for (int i = 1; i <= cells - 1; i++)
9      {
10         rectangle.setPosition(one_cell_size, 0);
11         window.draw(rectangle);
12         one_cell_size = one_cell_size + tmp;
13     }
14
15     one_cell_size = window.getSize().x / cells;
16     rectangle.rotate(-90);
17     for (int i = 1; i <= cells - 1; i++)
18     {
19         rectangle.setPosition(0, one_cell_size);
20         window.draw(rectangle);
21         one_cell_size = one_cell_size + tmp;
22     }
23 }
```

3.3. Obsługa „wydarzeń”

Każdorazowe kliknięcie myszką na planszę jest obsługiwane przez *sf::Event*. Po wykryciu kliknięcia, obliczana jest pozycja myszki i rysowany jest znak kółka w odpowiednim miejscu. Niestety program

nie jest odporny na błędy w postaci kilkukrotnego klikania myszką w jedno miejsce. W takim przypadku na planszy pojawi się więcej krzyżyków tak, jakby gra toczyła się dalej. Kod przedstawiony poniżej opisuje także obsługę zamknięcia okna z grą, co jest jednoznaczne z zakończeniem gry.

```
while (board.pollEvent(event))
{
    if (event.type == sf::Event::Closed)
        board.close();

    if (event.type == sf::Event::MouseButtonPressed)
    {
        if (event.mouseButton.button == sf::Mouse::Left)
        {
            sf::Vector2i position = sf::Mouse::getPosition(board);
            int column = position.x / one_cell_size;
            int row = position.y / one_cell_size;
            tab[row][column] = 'O';
            circle.setPosition(column * one_cell_size +
            (circle.getCharacterSize() * 0.1),
            row * one_cell_size + (circle.getCharacterSize() * -0.13));
            board.draw(circle);
            board.display();
        }
    }
}
```

4. Techniki AI

4.1. Wstęp

Ogólnie, wykorzystane techniki można przedstawić za pomocą grafów, gdzie każde kolejne rozgałęzienie pokazuje możliwe ruchy gracza i komputera. Aby wydedukować jaki ruch będzie najoptymalniejszy do wykonania przez komputer, musi on z zadanej pozycji obliczyć wszystkie możliwości i wybrać tę, która da mu najszybsze zwycięstwo. W tym celu wykorzystano strategię Minimax, która ma na celu maksymalizację szans na wygraną jednego użytkownika (w tym przypadku komputera) i minimalizację szans na wygraną przeciwnika (w tym przypadku gracza). Ze względu na to, że pełen algorytm Minimax, czyli rozważanie wszystkich przypadków i rozwijanie drzewa do końca byłby zbyt kosztowny i długotrwały, zastosowano funkcję heurystyczną oraz alfa-beta cięcia. Funkcja heurystyczna ma na celu jedynie oszacowanie najoptymalniejszego ruchu, nie gwarantuje przy tym, że będzie to ruch prawidłowy czy najlepszy. Natomiast alfa-beta cięcia pozwalają wyeliminować z analizy węzły nie mające wpływu na wartośc przodków. To znaczy, że jeżeli w jakiejś części grafu znaleziono jakąś lepszą wartość i jakąś gorszą wartość, to ten węzeł, który wskazuje gorszą wartość, zostanie porzucony i komputer nie będzie analizować dalej tej gałęzi. Techniki AI zostały zaimplementowane jako 3 funkcje: *rate_positions* - do liczenia i nadawania wag poszczególnym rozwiązaniom, *minimax_alpha_beta* - do wybierania najlepszej wagi oraz *best_ai_move* do wywoływania algorytmu Minimax i do zwracania najlepszego ruchu dla komputera. Funkcje te zostały przedstawione i opisane poniżej.

4.2. Funkcja *rate_positions* - do liczenia i nadawania wag poszczególnym rozwiązaniom

```
1  int rate_positions(int &number_of_cells, char **tab)
2  {
3      int position_rating = 0;
4      int tmp = 0;
5
6      for (int i=0; i < number_of_cells; i++)
7      {
8          tmp = 0;
9          for (int j=0; j < number_of_cells; j++)
10         {
11             if (tab[i][j] == 'X')
12                 tmp++;
13
14             else if (tab[i][j] == 'O')
15             {
16                 tmp = 0;
17                 break;
18             }
19         }
20
21         if (tmp)
22             position_rating += pow(10, tmp);
23
24
25     for (int i=0; i < number_of_cells; i++)
26     {
27         tmp = 0;
28         for (int j=0; j < number_of_cells; j++)
29         {
30             if (tab[j][i] == 'X')
31                 tmp++;
32
33             else if (tab[j][i] == 'O')
34             {
35                 tmp = 0;
36                 break;
37             }
38         }
39
40         if (tmp)
41             position_rating += pow(10, tmp);
42     }
43
44     tmp = 0;
```

```
45      for (int j=0; j < number_of_cells; j++)
46      {
47          if (tab[j][j] == 'X')
48              tmp++;
49
50          else if (tab[j][j] == 'O')
51          {
52              tmp = 0;
53              break;
54          }
55
56          if (tmp)
57              position_rating += pow(10, tmp);
58      }
59
60
61      for (int i = 0, j = number_of_cells - 1; i < number_of_cells; i++, j--)
62      {
63          tmp = 0;
64          if (tab[i][j] == 'X')
65              tmp++;
66
67          else if (tab[i][j] == 'O')
68          {
69              tmp = 0;
70              break;
71          }
72
73          if (tmp)
74              position_rating += pow(10, tmp);
75      }
76      return position_rating;
77      }
78  }
```

Funkcja *rate_positions* jest wywoływana w funkcji *minimax_alpha_beta* w dwóch przypadkach: po zakończeniu gry albo gdy głębokość osiągnie wartość 0, czyli przy ostatnim wywołaniu Minimaxa. W obydwu przypadkach komputer analizuje sytuację na planszy, kiedy wszystkie pola są zajęte. Zatem funkcja ta składa się z przeszukiwania kolejno rzędów, kolumn i dwóch przekątnych. Jeżeli w którejś z nich znajdzie ciągłość znaków „X" (czyli tych, którymi gra komputer), to nadaje takiemu rozwiązaniu odpowiednio dużą wagę. Jeżeli natomiast znajdzie rozwiązania, w których na przykład znaki „X" są rozdzielone znakiem „O" lub też rozwiązania, w których znaki „O" zachowują ciągłość w linii, to nie nadaje im wag.

4.3. Funkcja *minimax_alpha_beta* - do wybierania najlepszej wagi

```
1  int minimax_alpha_beta(char current_player, int depth, int a, int b,
2   int &number_of_cells, char **tab)
```

```
3  {
4      check_win(number_of_cells, tab);
5
6      if (winner != '-')
7      {
8          if (current_player == 'X')
9              return pow(10,8);              // max wartosc
10         else
11             return -pow(10,8);         // min wartosc
12     }
13
14     if (is_finished(number_of_cells, tab) || depth == 0)
15     {
16         if (current_player == 'X')
17             return rate_positions(number_of_cells, tab);
18         else
19             return -rate_positions(number_of_cells, tab);
20     }
21
22     int best_score;
23
24     if (current_player == 'X')
25     {
26         current_player = 'O';            // zamiana graczy
27         best_score = infinity;
28     }
29     else
30     {
31         current_player = 'X';
32         best_score = -infinity;
33     }
34
35     int tmp;
36     for (int i = 0; i < number_of_cells; i++)
37     {
38         for (int j = 0; j < number_of_cells; j++)
39         {
40             if (tab[i][j] == '-')
41             {
42                 if (current_player == 'X')
43                 {
44                     tab[i][j] = 'X';
45                     tmp = minimax_alpha_beta(current_player, depth-1, a,
46                     b, number_of_cells, tab);
47                     if (best_score < tmp)
48                         best_score = tmp;
49
```

```
50                      if (a < best_score)
51                          a = best_score;
52
53                      tab[i][j] = '-';
54                      if (a >= b)
55                          return best_score;
56                  }
57                  else
58                  {
59                      tab[i][j] = 'O';
60                      tmp = minimax_alpha_beta(current_player, depth-1, a,
61                      b, number_of_cells, tab);
62                      if (best_score > tmp)
63                          best_score = tmp;
64
65                      if (a > best_score)
66                          a = best_score;
67
68                      tab[i][j] = '-';
69                      if (a >= b)
70                          return best_score;
71                  }
72              }
73          }
74      }
75      return best_score;
76  }
```

Funkcja *minimax_alpha_beta* najpierw sprawdza czy funkcja już rozwinęła ścieżkę, w której doszło do zwycięstwa. Jeżeli tak, rozwiązaniom, w których zwyciężył „X" (komputer) przypisuje maksymalną wartość, a rozwiązaniom, w których zwyciężyło „O" (użytkownik), przypisuje minimalną wartość. Następnie, jeżeli gra jest skończona lub algorytm doszedł już do wskazanej głębokości, oceniana jest pozycja. Potem w każdorazowym wywołaniu funkcji, następuje zamiana gracza, którego ruch jest rozważany. Następnie, algorytm przechodzi po komórkach tablicy i w wolne miejsca wpisuje odpowiedni znak po to, by rekurencyjnie znów się wywołać, z głębokością pomniejszoną o 1. W ten sposób algorytm dochodzi do końcowej głębokości (do pewnego rozwinięcia drzewa), z czego każda rozważana możliwość zagrania ma swoją wartość. Na koniec algorytm stosuje alfa-beta cięcia, to znaczy, że będąc na końcowej głębokości, wraca do rodzica, wybierając najlepszą ścieżkę i odrzucając inne rozwiązania.

4.4. Funkcja best_ai_move - do wywoływania algorytmu Minimax i do zwracania pozycji najlepszego ruchu dla komputera

```
1  std::pair<int, int> best_ai_move(int depth, int &number_of_cells, char **tab)
2  {
3      int best_score = minus_infinity;
4      int tmp;
```

```
5       int set_i;
6       int set_j;
7
8       for (int i = 0; i < number_of_cells; i++)
9       {
10          for (int j = 0; j < number_of_cells; j++)
11          {
12              if (tab[i][j] == '-')
13              {
14                  tab[i][j] = 'X';
15                  tmp = minimax_alpha_beta('X', depth, -infinity,
16                  infinity, number_of_cells, tab);
17                  tab[i][j] = '-';
18                  if (tmp > best_score)
19                  {
20                      best_score = tmp;
21                      set_i = i;
22                      set_j = j;
23                  }
24              }
25          }
26      }
27
28      if (set_i < number_of_cells && set_j < number_of_cells)
29          tab[set_i][set_j] = 'X';
30
31      return std::make_pair(set_i, set_j);
32  }
```

Funkcja ta przechodzi kolejno po komórkach planszy i gdy napotka puste pole, wpisuje tam znak „X", po czym wywołuje funkcje *minimax_ alpha_ beta*, która rozbudowuje drzewo i wybiera najlepszą wagę, którą zapisuje do zmiennej *tmp*. Po cofnięciu wprowadzonej zmiany do komórki planszy, funkcja ta porównuje zmienną *tmp* z inną ustaloną wartością (minimalną). Jeżeli znajdzie pozycję większą od ustalonej, wpisuje i zwraca w którą komórkę komputer wykona ruch.

5. Podsumowanie i wnioski

1. SFML jest bogato wyposażoną biblioteką, którą wykorzystuje się do tworzenia na przykład graficznych interfejsów czy gier. Nie oferuje jednak gotowych rozwiązań do tworzenia guzików, które byłyby przydatne do komunikacji z użytkownikiem. Zamiast tego, w projekcie wykorzystano w tym celu terminal.

2. W programie znajduje się niedopatrzenie, które umożliwia użytkownikowi wpisanie znaku nieskończenie wiele razy w tę samą komórkę planszy.

3. Program działa poprawnie. Dla każdego testowanego rozmiaru planszy zaimplementowane techniki AI blokują wygraną użytkownika i dążą do wygranej komputera.

4. Przed rozpoczęciem należy ustawić poziom trudności gry. Jest to jednoznaczne z ustawieniem głębokości, do jakiej komputer będzie liczyć różne warianty. Optymalny wybór poziomu trudności zależy od rozmiaru planszy. Dla planszy 3x3 i głębokości 5, ruch komputera następuje praktycznie od razu, natomiast dla planszy 5x5 i tej samej głębokości, na ruch komputera trzeba poczekać około 50 sekund.

5. Jest jeszcze wiele rzeczy do dopracowania bądź zaimplementowania w omawianym projekcie, na przykład: dodanie możliwości wyboru znaku przez użytkownika, lepsze oznaczenie wygranej z wykorzystaniem graficznego interfejsu, czy na przykład możliwość definiowania liczby tych samych znaków w linii potrzebnej do wygranej.

6. Literatura

- https://www.sfml-dev.org/ (dostęp: 8.06.2022)
- https://en.wikipedia.org/wiki/Minimax (dostęp: 8.06.2022)
- Notatki z wykładów