

Projektowanie algorytmów i metody sztucznej inteligencji

Projekt
07.06.2021

PROJEKT 3
GRA - SZACHY

Autor

MIKOŁAJ ZAPOTOCZNY
(252939)

Prowadzący

Mgr inż. Marta Emirsajłowa

1 Wprowadzenie

Zrealizowałem zadanie na maksymalną ocenę bdb. Zdecydowałem się napisać proste szachy, które jednak nie mają wszystkich funkcjonalności. Metody sztucznej inteligencji w grach nie były jeszcze realizowane na wykładzie, stąd mój program jest bardzo ubogi w sztuczną inteligencję. Próbowałem podpiąć stockfische, ale okazało się to trudniejsze niż myślałem. Szachy są bardzo skomplikowaną grą i na moim poziomie trudno jest zaimplementować wszystkie zasady, ale mam nadzieję, że to co zrobiłem wystarczy.

Link do GitHuba: [<KOD PROGRAMU>](#)

2 Opis programu

2.1 Kompilacja

Mój program zawiera warstwę graficzną, więc kompilacja nie jest taka prosta. Używam tutaj SFML, aby zapewnić interakcję z obrazem, stąd kolejne kroki do uruchomienia programu są następujące:

- `g++ -c main.cpp -I/include`
- `g++ main.o -o sfml-app -L/lib -lsfml-graphics -lsfml-window -lsfml-system`
- `./sfml-app`

Na początku tworzymy plik o rozszerzeniu `.o`, a dopiero potem poddajemy go linkowaniu, co jest wymagane przez SFML.

2.2 Warstwa graficzna

Po uruchomieniu programu wyświetla się plansza do gry wraz z bierkami do gry. Plansza została w programie podzielona na 64 pola oraz każde pole ma swoją wartość, czyli jest jedną z figur, lub pustym polem. Plansza oraz bierki do gry są obrazkami pobranymi z Internetu. Szczególnie cenny jest tutaj obrazek bierek szachowych, gdyż są one wycięte bez żadnego tła, co bardzo ułatwia pisanie programu. Ten obrazek jest bardzo często wykorzystywany, ale nie ma co się temu dziwić. Z specjalnych rzeczy w grze w szachy udało mi się zrealizować roszadę, czyli wtedy kiedy król zamienia się miejscami z wieżą. Nie udało mi się zrealizować takich rzeczy jak: bicie w przelocie, czy też dojście pionkiem do pola przemiany - okazało się to być zbyt skomplikowane.

Możemy "złapać" każdą bierkę i przenieść ją w dowolne pole. Jeżeli z kimś gramy to musimy być uważni. Udało mi się opanować bicie, tzn, gdy jedną bierką najeżdżamy na inną i puszczamy przycisk myszy to wtedy bita bierka po prostu znika, co symuluje bicie w prawdziwej grze.



2.3 Kod

```

1  #include <SFML/Graphics.hpp>
2  #include <time.h>
3  #include <iostream>
4  using namespace sf;
5
6  int rozmiar = 56; // rozmiar potrzebny do
wydzielania odpowiedniej czesci obrazka
7  Vector2f zmiana(28,28); // wektor potrzebny do
zapamietywania pozycji
8
9  Sprite figura[32]; // ilosc figur na planszy na
poczatku
10 std::string pozycja=""; // string na zapis pozycji
11
12 int plansza[8][8] =
13     {-1,-2,-3,-4,-5,-3,-2,-1,
14      -6,-6,-6,-6,-6,-6,-6,-6,
15       0, 0, 0, 0, 0, 0, 0, 0,
16       0, 0, 0, 0, 0, 0, 0, 0,
17       0, 0, 0, 0, 0, 0, 0, 0,
18       0, 0, 0, 0, 0, 0, 0, 0,
19       6, 6, 6, 6, 6, 6, 6, 6,
20       1, 2, 3, 4, 5, 3, 2, 1}; // tablica z zapisem
pozycji na poczatku
21
22 std::string zapisruchow(Vector2f pole) // funkcja potrzebna
do wypisywania ruchow
23 {
24     std::string ruch = "";
25     ruch += char(pole.x/rozmiar+97);
26     ruch += char(7-pole.y/rozmiar+49);
27     return ruch;
28 }

```

```

29
30 Vector2f bicie(char a,char b) // funkcja odpowiedzialna za bicie
    figur
31 { // a i b to wspolrzedne figury,
    ktora bierze udzial przy biciu
32     int x = int(a) - 97;
33     int y = 7-int(b)+49;
34     return Vector2f(x*rozmiar,y*rozmiar);
35 }
36
37 void move(std::string str) // funkcja odpowiada za
    poruszanie figurami
38 {
39     Vector2f oldPos = bicie(str[0],str[1]); // podmieniamy stare
    wartosci na nowe po biciu
40     Vector2f newPos = bicie(str[2],str[3]);
41
42     for(int i=0;i<32;i++) // aktualna pozycja
43         if (figury[i].getPosition()==newPos) figury[i].setPosition
    (-100,-100);
44
45     for(int i=0;i<32;i++) // poprzednia pozycja
46         if (figury[i].getPosition()==oldPos) figury[i].setPosition(newPos);
47
48     if (str=="e1g1") if (pozycja.find("e1")==-1) move("h1f1"); //
    implementacja roszady, czyli wtedy kiedy krol nie poruszyl sie ze
    swojego pola
49     if (str=="e8g8") if (pozycja.find("e8")==-1) move("h8f8"); // to
    wtedy moze zamienic sie miejscami z wieza, tzn poruszyc sie o dwa pola
50     if (str=="e1c1") if (pozycja.find("e1")==-1) move("a1d1");
51     if (str=="e8c8") if (pozycja.find("e8")==-1) move("a8d8");
52 }
53
54 void loadPosition() // funkcja odpowiada za to, aby
    kazda figura znajdowala sie tam gdzie ma byc, tzn zeby bylo
55 { // osiem pionkow kazdego koloru i
    bierze z obrazka odpowiedni fragment i go wrzuca na dene miejsce
56     int k=0;
57     for(int i=0;i<8;i++)
58     for(int j=0;j<8;j++)
59     {
60         int n = plansza[i][j];
61         if (!n) continue;
62         int x = abs(n)-1;
63         int y = n>0?1:0;
64         figury[k].setTextureRect( IntRect(rozmiar*x,rozmiar*y,rozmiar,
    rozmiar) ); // bierzemy te czesc obrazka, ktora nas interesuje
65         figury[k].setPosition(rozmiar*j,rozmiar*i);
66         k++;
67     }
68
69     for(int i=0;i<pozycja.length();i+=5)
70         move(pozycja.substr(i,4)); // zmiana
    dla kazdego kolejnego ruchu
71 }
72
73
74 int main()
75 {
76     RenderWindow window(VideoMode(504, 504), "PAMSI P3 - Szachy!"); //

```

```

do pokazywania planszy szachowej

77
78     Texture t1,t2;                                     //
dwie tekstury odpowiadajace za plansze i za bierki
79     t1.loadFromFile("obrazy/figury.png");
80     t2.loadFromFile("obrazy/plansza.png");
81
82     for(int i=0;i<32;i++) figury[i].setTexture(t1);      //
zrobienie miejsca na figury na planszy
83     Sprite sBoard(t2);                                  //
zmienna dla planszy

84
85     loadPosition();                                     // zaladowanie pozycji
86
87     bool isMove=false;                                  // czy wykonujemy ruch czy nie
88     float dx=0, dy=0;                                   // wspolrzedne
89     Vector2f oldPos,newPos;                              // nowa i stara pozycja
90     std::string str;                                     // do zapisu ruchow
91     int n=0;
92
93     while (window.isOpen())
94     {
95         Vector2i pos = Mouse::getPosition(window) - Vector2i(zmiana);    //
wyswietlanie nowej pozycji po kontakcie myszki z oknem
96
97         Event e;                                         // Jakas akcja, u mnie zwiazana
z myszka, ktora dzieje sie w aplikacji
98         while (window.pollEvent(e))
99         {
100             if (e.type == Event::Closed)               // Koniec programu gdy okno
zamkniete
101                 window.close();
102
103             if (e.type == Event::KeyPressed) // Cofniecie ruchu
104             if (e.key.code == Keyboard::BackSpace)
105             {
106                 pozycja.erase(pozycja.length()-6,5);
107                 loadPosition();
108             }
109
110             if (e.type == Event::MouseButtonPressed)    // Lapanie i
przenoszenie figur
111             if (e.key.code == Mouse::Left)              // z uzyciem lewego
klawisza myszki
112             for(int i=0;i<32;i++)                      // sprawdzanie
wszystkich figur
113             if (figury[i].getGlobalBounds().contains(pos.x,pos.y)) // zmiana
pozycji tej konkretnej figury
114             {
115                 isMove=true; n=i;
116                 dx=pos.x - figury[i].getPosition().x;
117                 dy=pos.y - figury[i].getPosition().y;
118                 oldPos = figury[i].getPosition();
119             }
120
121             if (e.type == Event::MouseButtonReleased)    // Gdy puszcza
klawisz myszki
122             if (e.key.code == Mouse::Left)
123             {
124                 isMove=false;

```

```

125         // Konczymy ruch
126         Vector2f p = figury[n].getPosition() + Vector2f(rozmiar/2,
rozmiar/2); // Wyrownanie do kratek planszy
127         newPos = Vector2f( rozmiar*int(p.x/rozmiar), rozmiar*int(p.y/
rozmiar) ); // Zapisujemy nowa pozycje
128         str = zapisruchow(oldPos)+zapisruchow(newPos);
// Aktualizacja spisu ruchow
129         std::cout<<str<<std::endl;
// Wyświetlenie ruchu
130         move(str);
// Wykonanie ruchu (bicie itp)
131         if (oldPos!=newPos) pozycja+=str+" ";
132         figury[n].setPosition(newPos);
133     }
134     if (isMove) figury[n].setPosition(pos.x-dx,pos.y-dy);
135
136     window.clear(); // pokazanie w oknie
aplikacji wszystkiego
137     window.draw(sBoard);
138     for(int i=0;i<32;i++) figury[i].move(zmiana);
139     for(int i=0;i<32;i++) window.draw(figury[i]); window.draw(figury[n]);
140     for(int i=0;i<32;i++) figury[i].move(-zmiana);
141     window.display();
142 }
143
144     return 0;
145 }
146
147

```

3 Opis programu

Zacznijmy od opisu funkcji main. Na początku tworzymy okno, podajemy jego wymiary oraz tytuł tego okna. Potem wczytujemy obrazki figur i planszy jako tekstury, na których potem będziemy operować. Każdą z tekstur przygotowujemy do jej roli w programie.

Następnie używamy funkcji loadPosition, która będzie opisana potem, ale teraz niech nam wystarczy, że ta funkcja wgrywa aktualną pozycję, w tym miejscu w programie jest to ustawienie początkowe. Potem deklarujemy potrzebne zmienne i przechodzimy do pętli while, która działa gdy okno aplikacji jest otwarte.

W pętli while są zawarte operacje wykonywane na myszce, tzn, że gdy myszka będzie mieć kontakt z programem to coś się wykona. Tworzymy zmienną typu Event, czyli jakieś zdarzenie i implementujemy te różne zdarzenia.

Po pierwsze gdy okno programu jest zamknięte to kończy się działanie programu. W tym celu używamy poprostu funkcji: window.close().

Następną funkcjonalnością jest możliwość cofnięcia ruchu przy pomocy klawisza Back-Space, co powoduje wyświetlenie pozycji na szachownicy sprzed nieudanego ruchu.

Następnie jest to co najważniejsze, czyli łapanie i przenoszenie figur za pomocą lewego klawisza myszki. za każdym razem sprawdzamy wszystkie figury, bo nie zawsze ruch oznacza przesunięcie tylko jednej z nich, na co przykładem jest roszada lub bicie. Następnie zmieniamy pozycję oraz zapisujemy poprzednią.

Potem jest funkcja co zrobić gdy klawisz myszy zostanie puszczony. Ustawiamy brak ruchu. W tym miejscu następuje wyrównanie bierek szachowych do kratek planszy. Zapisujemy nową pozycję. Zapisujemy ruch, aby go potem wypisać i wykorzystać do ewentualnego

cofnięcia. Wykonanie ruchu oraz ewentualna pomoc przy zdarzeniach niestandardowych, np bicie lub roszada.

Na koniec pozostaje nam kolejno wyświetlać tablicę oraz każdą bierkę korzystając z informacji o tym jaka jest teraz pozycja.

Przejdźmy teraz do funkcji zadeklarowanych na początku programu. Najpierw są zadeklarowane zmienne, np tablica odpowiadająca za przypisanie numeru do każdego pola i każda operacja matematyczna opiera się właśnie na tej planszy. Jest to taki wirtualny odpowiednik tej planszy, którą widzimy, tylko, że na niej wykonywane są odpowiednie działania.

Poprzez odpowiednio wyliczone działania dobrze funkcjonuje funkcja do zapisu kolejnych ruchów : zapisruchow. Po prostu tworzy string, który potem będzie wypisany. Funkcja bicie odpowiada za znikanie jednej figury i zastępowanie ją inną, gdy użyjemy ją dwa razy, co jest wykorzystane w funkcji: move. W te funkcji są właśnie operacje odpowiedzialne za bicie, oraz za roszadę, która jest dodatkową funkcjonalnością tego programu. Niestety nie udało mi się zaimplementować bicia w przelocie oraz zamiany pionka w hetmana w polu przemiany. Dodatkowo funkcja zapisuje aktualne ustawienie figur na planszy dla aktualnej i poprzedniej pozycji.

Ostatnią funkcją do omówienia jest funkcja loadPosition, która daje do programu aktualną pozycję przeniesioną na warstwę graficzną, gdzie dla obrazka z bierkami bierze tylko tę część tego obrazka, która nas interesuje w danym momencie. Na końcu jest jeszcze zmiana pozycji wizualnie po każdym ruchu.

4 Wnioski

- Napisanie tych ok. 150 linijek było bardziej czasochłonne i wymagające niż wszystkie poprzednie projekty.
- Nie wykorzystałem żadnych metod sztucznej inteligencji, bo nie było tego jeszcze prawie wcale na wykładnie.
- SFML teoretycznie powinien działać tylko pod windowsem, ale okazało się, że całkiem nieźle działa też pod Linuxem.
- Jest wiele przydatnych funkcji wbudowanych w przestrzeń nazw sf, co bardzo upraszcza pisanie kodu dla tego typu gier.
- Widzimy, że szachy działają, więc zadanie zostało zrealizowane choć w bardzo uproszczonej wersji.

5 Źródła informacji

- <https://forum.pclab.pl/topic/436933-szachy-c/>
- https://gitlab.com/wojton/szachy_cpp
- <https://forum.pasja-informatyki.pl/525513/atak-figur-szachy-c>
- https://www.youtube.com/watch?v=_4EuZI8Q8cs