

Programowanie obiektowe – raport projekt 1 C++

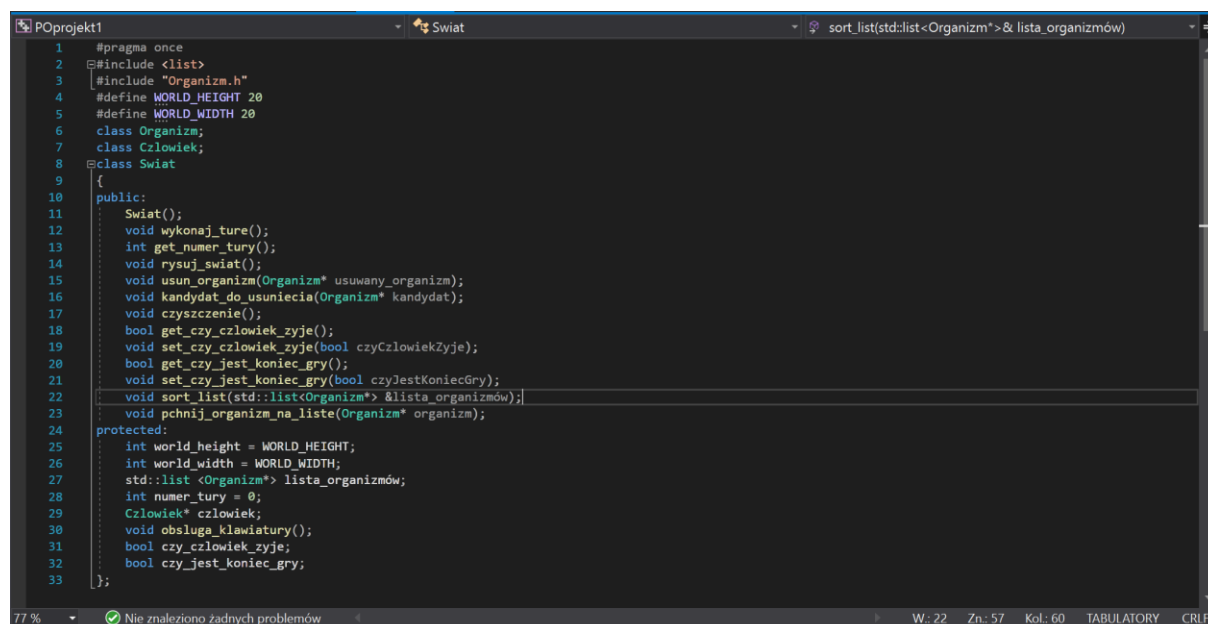
W poniższym dokumencie opiszę pracę, którą wykonałam podczas projektu pt. „Wirtualny świat” w ramach przedmiotu Programowanie obiektowe.

To mój pierwszy raport, więc mam nadzieję, że spełni on wymagania dotyczące opisu dokonanych działań i zrzutów ekranu zawierających kod projektu.

A więc zaczynając, projekt pt. „Wirtualny świat” stanowił dla mnie pewnego rodzaju wyzwanie, ale i krok milowy. Cieszę się, że mogłam wcześniej poznać podstawy programowania obiektowego i podszkolić swoje umiejętności podczas wykonywania zadań laboratoryjnych, ponieważ ułatwiło to i usprawniło znacząco pracę.

Gdy rozpoczęłam pracę nad projektem to czułam się w nim mocno zagubiona. Instrukcja wydawała się być mocno skomplikowana i zawiła. Na szczęście, wraz z upływem czasu i pogłębiania wiedzy, powoli zaczynałam rozumieć „jak to w ogóle ma działać”.

Moją pierwszą trudnością, na jaką napotkałam było stworzenie klasy abstrakcyjnej, która zawierała inną klasę abstrakcyjną, która miała wpływ na klasę nadrzędną i na odwrót. Okazało się, że należy w tych dwóch plikach nagłówkowych załączyć „**class Swiat;**” i odpowiednio „**class Organizm;**”.



```
1 #pragma once
2 #include <list>
3 #include "Organizm.h"
4 #define WORLD_HEIGHT 20
5 #define WORLD_WIDTH 20
6 class Organizm;
7 class Czlowiek;
8 class Swiat
9 {
10 public:
11     Swiat();
12     void wykonaj_ture();
13     int get_numer_tury();
14     void rysuj_swiat();
15     void usun_organizm(Organizm* usuwany_organizm);
16     void kandydat_do_usuniecia(Organizm* kandydat);
17     void czyszczenie();
18     bool get_czy_czlowiek_zyje();
19     void set_czy_czlowiek_zyje(bool czyCzlowiekZyje);
20     bool get_czy_jest_koniec_gry();
21     void set_czy_jest_koniec_gry(bool czyJestKoniecGry);
22     void sort_list(std::list<Organizm*> &lista_organizmow);
23     void pchnij_organizm_na_liste(Organizm* organizm);
24 protected:
25     int world_height = WORLD_HEIGHT;
26     int world_width = WORLD_WIDTH;
27     std::list<Organizm*> lista_organizmow;
28     int numer_tury = 0;
29     Czlowiek* czlowiek;
30     void obsluga_klawiatury();
31     bool czy_czlowiek_zyje;
32     bool czy_jest_koniec_gry;
33 };
```

Następnie zdałam sobie sprawę, że żeby wygodnie operować na obiektach klasy **Zwierze** i **Roslina**, potrzebuję je trzymać w jakimś pojemniku na dane, który będzie elastyczny i najlepiej posiadał jakieś wbudowane funkcje do np. usuwania elementów z automatycznym uaktualnieniem liczby zawartych w tym pojemniku elementów.

Zastanawiałam się nad **<vector>** i **<list>**. Zdecydowałam się na trzymanie obiektów w **<list>**, ponieważ na przedmiocie Algorytmy i struktury danych na projekcie 2. i 3. pisaliśmy już swoje listy od zera i byłam zaznajomiona z ich działaniem. Spróbowałam teraz użyć gotowej listy z biblioteki. Nigdy wcześniej nie używałam wbudowanych szablonów z biblioteki stl, ale czytając dokumentację poradziłam sobie z jej zastosowaniem.

W projekcie użyłam wbudowanych funkcji: **push_back()**, **begin()**, **end()** i **erase()**. Podczas pisania użyłam także **reverse()**, ale w ostatecznej wersji zrezygnowałam z niej i rozwiązałam problem w inny sposób.

Miałam problem z usuwaniem organizmów. Zrobienie go poprawnie zajęło mi sporo czasu i nerwów, ponieważ wychodziłam poza zakres listy. Okazało się, że inkrementowałam iterator po elementach listy jeszcze po usunięciu danego elementu. Od teraz już wiem, na co zwracać uwagę.

W projekcie utworzyłam klasę Świat (Swiat) zarządzającą rozgrywką i organizmami. Zawiera ona m.in. metody, np: **wykonaj_ture()**, **rysuj_swiat()** oraz pola.

Utworzyłam również abstrakcyjną klasę **Organizm**.

Jej podstawowe **pola (protected)**:

- **siła**
- **inicjatywa**
- **położenie**, czyli struktura **Pozycja_t** z dwoma polami, przechowującymi koordynaty obiektu x i y
- **świat** - referencja do świata w którym znajduje się organizm.

Podstawowe **metody** klasy **Organizm**:

- **akcja()** → określa zachowanie organizmu w trakcie tury,
- **kolizja()** → określa zachowanie organizmu w trakcie kontaktu/zderzenia z innym organizmem,
- **rysowanie()** → powoduje narysowanie symbolicznej reprezentacji organizmu. **Override'uję je w klasach Zwierze i Roslina.**

Klasa **Organizm** jest abstrakcyjna, dziedziczą po niej dwie kolejne klasy czyli **Zwierze** i **Roslina**. Odkryłam, że istnieje coś takiego jak **enum** i użyłam go trzykrotnie, bardzo wygodnie się go używa. Z początku w każdym pliku .cpp miałam takie same **#define'y**, co wyglądało kiepsko i można było się łatwo pogubić. W przypadku wykorzystania enuma trudniej jest się pomylić i zrobić jakiś dość głupi błąd, np. w jednym z plików zamienić wartości definów lewo z prawo (co spowodowało tymczasowe błędy w poruszaniu się zwierząt).

Poniżej wstawiam plik nagłówkowy **Organizm.h**:

```

1  #pragma once
2  #include <iostream>
3  #include <list>
4  #include "Swiat.h"
5  #include "Organizm.h"
6  using namespace std;
7  class Swiat;
8  class Organizm
9  {
10 public:
11     enum class Kierunek
12     {
13         GORA,
14         DOL,
15         LEWO,
16         PRAWO,
17         BRAK_KIERUNKU
18     };
19     enum class Skutek_kolizji
20     {
21         ROZMNAZANIE,
22         SMIERC_WYKONUJACEGO_AKCJE,
23         SMIERC_INNEGO,
24         ODGONENIE,
25         UCIECZKA,
26         SMIERC_OBU
27     };
28     enum class TypOrganizmu
29     {
30         CZLOWIEK,
31         OMCA,
32         WILK,
33         LIS,
34         ZOLW,
35         ANTYLOPA,
36         TRAMA,
37         MLECZ,
38         GUARANA,
39         WILCZE_JAGODY,
40         BARSZCZ_SOSNOWSKIEGO
41     };
42 };

```

```

40     BARSZCZ_SOSNOWSKIEGO
41 };
42 TypOrganizmu GetTypOrganizmu();
43 void SetTypOrganizmu(TypOrganizmu typOrganizmu);
44
45 virtual void akcja(std::list<Organizm*> & lista_organizmow) = 0;
46 virtual int kolizja(Organizm* organizm_z_ktorym_kolizja, std::list<Organizm*> lista_organizmow) = 0;
47 virtual void rysowanie() = 0;
48
49 virtual int getX();
50 virtual int getY();
51 virtual int setX(int a);
52 virtual int setY(int a);
53 virtual int getSila();
54 virtual int getPoprzedniX() = 0;
55 virtual int getPoprzedniY() = 0;
56 virtual int getInicjatywa();
57 virtual int setsila(int a);
58 virtual int setInicjatywa(int a);
59 virtual void set_tura_urodzenia(int turaurodzenia);
60 virtual int get_tura_urodzenia();
61 virtual string TypOrganizmuToString() = 0;
62 void OrganizmToString();
63 virtual bool czy_zwierze() = 0;
64 bool do_usuniecia = false;
65 protected:
66     int sila, inicjatywa, turaUrodzenia;
67     Swiat* swiat;
68     struct Pozycja_t
69     {
70         int pozycja_x;
71         int pozycja_y;
72     } koordynaty;
73     TypOrganizmu typOrganizmu;
74     Organizm(Swiat* swiat, TypOrganizmu typOrganizmu, int sila, int inicjatywa, int x, int y, int turaUrodzenia);
75 };

```

W klasie **Zwierze** zaimplementowałam wspólne dla wszystkich/większości zwierząt zachowania, przede wszystkim:

- podstawową formę ruchu w metodzie **akcja()** → każde typowe zwierzę w swojej turze przesuwa się na wybrane losowo, sąsiednie pole,
- rozmnażanie w ramach metody **kolizja()** → przy kolizji z organizmem tego samego gatunku nie dochodzi do walki, oba zwierzęta pozostają na swoich miejscach, koło nich pojawia się trzecie zwierzę, tego samego gatunku.

```

1  #pragma once
2  #include "Organizm.h"
3  class Zwierze : public Organizm
4  {
5  public:
6      void akcja(std::list<Organizm*>& lista_organizmow) override;
7      int kolizja(Organizm* organizm_z_ktorym_kolizja, std::list<Organizm*> lista_organizmow) override;
8      bool czy_zwierze() override;
9      virtual void rysowanie() = 0;
10     virtual int getPoprzedniX();
11     virtual int getPoprzedniY();
12     virtual int setPoprzedniX(int a);
13     virtual int setPoprzedniY(int a);
14     void Rozmnazanie(std::list<Organizm*> lista_organizmow, Organizm* organizm);
15 protected:
16     struct Pozycja_poprzednia_t
17     {
18         int poprzedni_x;
19         int poprzedni_y;
20     };
21     bool czy_mozesz_rozmnazac = true;
22     Zwierze(Swiat* swiat, TypOrganizmu typOrganizmu, int sila, int inicjatywa, int x, int y, int turaUrodzenia);
23     bool czy_mozesz_wykonac_ruch(int pos_x, int pos_y, std::list<Organizm*> lista_organizmow);
24 };

```

Poniżej załączam przykładowy plik .h i .cpp jednego z pięciu zwierząt. W plikach .cpp zmieniam kolor tekstu w terminalu.

```

1  #pragma once
2  #include "Zwierze.h"
3  class Owca : public Zwierze
4  {
5  public:
6      Owca(Swiat* swiat, int x, int y, int turaUrodzenia);
7      string TypOrganizmuToString() override;
8      void rysowanie() override;
9  };

```

```

1  #include "Owca.h";
2  #include "Kolory.h"
3  #define OWCA_SILA 4
4  #define OWCA_INICJATYWA 4
5  using namespace std;
6
7  Owca::Owca(Swiat* swiat, int x, int y, int turaurodzenia)
8      :Zwierze(swiat, TypOrganizmu::OWCA, OWCA_SILA, OWCA_INICJATYWA, x, y, turaUrodzenia)
9  {}
10 string Owca::TypOrganizmuToString()
11 {
12     return "Owca";
13 }
14 void Owca::rysowanie()
15 {
16     HANDLE hOut;
17     hOut = GetStdHandle(STD_OUTPUT_HANDLE);
18     SetConsoleTextAttribute(hOut, BROWN);
19     cout << "O";
20     SetConsoleTextAttribute(hOut, WHITE);
21 }

```

Stworzyłam również klasę **Czlowiek**, która **dziedziczy po klasie Zwierze**. Nie posiada on własnej inteligencji. Sterowany jest przez gracza strzałkami na klawiaturze, prawo, lewo, góra, dół. Do obsługi człowieka, czyli wczytywania inputu od gracza użyłam metody **obsługa_klawiatury()** (**protected w Swiat.h**) i stworzyłam plik nagłówkowy **Klawiatura.h** przechowujący key symbols klawiszy funkcyjnych.

```

1 #pragma once
2 #include "Zwierze.h"
3 #class Czlowiek : public Zwierze
4 {
5 public:
6     Czlowiek(Swiat* swiat, int x, int y, int turadrodzenia);
7     string TypOrganizmuToString() override;
8     void akcja(std::list<Organizm*>& lista_organizmow) override;
9     void rysowanie() override;
10    class Umiejetnosc
11    {
12    public:
13        Umiejetnosc();
14        bool get_czy_umiejetnosc_aktywna();
15        void set_czy_umiejetnosc_aktywna(bool czy_aktywna);
16        bool get_czy_mozna_aktwowac();
17        void set_czy_mozna_aktwowac(bool czy_mozna_aktwowac);
18        int get_czas_trwania_umiejetnosci();
19        void set_czas_trwania(int czastrwania);
20        int get_czas_aktywacji();
21        void set_czas_aktywacji(int dostepny_czas_aktywacji);
22        void uaktualnij();
23        void aktywuj_umiejetnosc();
24        void dezaktywuj_umiejetnosc();
25    protected:
26        bool czy_aktywna;
27        bool czy_mozna_aktwowac;
28        int czas_trwania;
29        int dostepny_czas_aktywacji;
30    };
31    Kierunek get_kierunek_ruchu();
32    void set_kierunek_ruchu(Kierunek kierunekruchu);
33    Umiejetnosc* GetUmiejetnosc();
34    protected:
35        Kierunek kierunek_ruchu;
36        Umiejetnosc* umiejetnosc;
37        void szybkość_antylopy();
38        void przestaw_koordynaty();
39 };

```

Udało mi się zrealizować projekt na wymagania na 4 pkt, więc projekt nie posiada klawisza funkcyjnego Save. Projekt posiada **klawisze funkcyjne: X – wyjście z gry, U – aktywacja specjalnej umiejętności człowieka i Enter, czyli przejście do kolejnej tury**, co jest jednoznaczne z utratą ruchu człowieka.

Człowiek jest tylko jeden i się nie rozmnaża. Jego specjalną umiejętnością jest szybkość antylopy (mój indeks: 184592, (ostatnia cyfra indeksu) mod 5 == 2, 2 to szybkość antylopy). Po włączeniu tej umiejętności klikając U, człowiek zyskuje umiejętność. Jej czas trwania wynosi 5 tur, a możliwość ponownej aktywacji specjalnej umiejętności wynosi 5 tur po wygaśnięciu uprzednio włączonej specjalnej umiejętności.

Zaimplementowałam 5 klas zwierząt, dziedziczących po Zwierze: Owca, Wilk, Lis, Zolw i Antylopa.

W klasie Roślina zaimplementowałam wspólne dla wszystkich/większości roślin zachowania, przede wszystkim: • symulacja rozprzestrzeniania się rośliny w metodzie akcja() → z pewnym prawdopodobieństwem każda z roślin może „zasiać” nową roślinę tego samego gatunku na losowym, sąsiednim polu. Wszystkie rośliny mają zerową inicjatywę

Zaimplementowałam 5 klas roślin, dziedziczących po Roslina: trawa, mlecch, guarana, wilcze jagody i barszcz Sosnowskiego.

```

POprojekt1 Roslina
1 #pragma once
2 #include "Organizm.h"
3 class Roslina : public Organizm
4 {
5 public:
6 void akcja(std::list<Organizm*>& lista_organizmow) override;
7 int kolizja(Organizm* organizm_z_kolizja, std::list<Organizm*> lista_organizmow) override;
8 bool czy_zwierze() override;
9 virtual int getPoprzedniX();
10 virtual int getPoprzedniY();
11 void rysowanie() = 0;
12 protected:
13 Roslina(Swiat* swiat, TypOrganizmu typOrganizmu, int sila, int inicjatywa, int x, int y, int turaUrodzenia);
14 void Rozprzestrzezanie(std::list<Organizm*> lista_organizmow);
15 bool czy_moze_wykonac_ruch(int pos_x, int pos_y, std::list<Organizm*> lista_organizmow);
16 };

```

Przykładowa roślina ze specjalną kolizją (pliki .h i .cpp):

```

POprojekt1 Guarana
1 #pragma once
2 #include "Roslina.h"
3 class Guarana : public Roslina
4 {
5 public:
6 Guarana(Swiat* swiat, int x, int y, int turaUrodzenia);
7 string TypOrganizmuToString() override;
8 void rysowanie() override;
9 int kolizja(Organizm* organizm_z_kolizja, std::list<Organizm*> lista_organizmow) override;
10 };

```

```

POprojekt1 Guarana kolizja(Organizm* organizm, std::list<Organizm*> lista_organ
1 #include "Guarana.h";
2 #include "kolizja.h"
3 #define GUARANA_SILA 0
4 #define GUARANA_INICJATYWA 0
5 using namespace std;
6
7 Guarana::Guarana(Swiat* swiat, int x, int y, int turaUrodzenia)
8 : Roslina(swiat, TypOrganizmu::GUARANA, GUARANA_SILA, GUARANA_INICJATYWA, x, y, turaUrodzenia)
9 {}
10
11 string Guarana::TypOrganizmuToString()
12 {
13 return "Guarana";
14 }
15
16 void Guarana::rysowanie()
17 {
18 HANDLE hOut;
19 hOut = GetStdHandle(STD_OUTPUT_HANDLE);
20 SetConsoleTextAttribute(hOut, MAGENTA);
21 cout << "a";
22 SetConsoleTextAttribute(hOut, WHITE);
23 }
24
25 int Guarana::kolizja(Organizm* organizm, std::list<Organizm*> lista_organizmow)
26 {
27 if (this->TypOrganizmuToString() == organizm->TypOrganizmuToString())
28 {
29 organizm->setX(organizm->getPoprzedniX());
30 organizm->setY(organizm->getPoprzedniY());
31 cout << "rozmazanie\n";
32 Sleep(1000);
33 return (int)Skutek_kolizji::ROZMAZANIE;
34 }
35 else
36 {
37 if (this->sila < organizm->getSila())
38 {
39 organizm->setSila(organizm->getSila() + 1);
40 cout << organizm->TypOrganizmuToString() << " x[" << organizm->getX() << "] y[" << organizm->getY() << "] sila[" << organizm->getSila() << "] zaleksza sila o 1, teraz jego sila wynosi: " << organizm->getSila() << endl;
41 cout << "miera roslina: " << this->TypOrganizmuToString() << endl;
42 Sleep(1000);
43 return (int)Skutek_kolizji::SMIERC_INNEGO;
44 }
45 else if (this->sila > organizm->getSila())
46 {
47 cout << "miera organizm wykonujacy akcje: " << organizm->TypOrganizmuToString() << endl;
48 Sleep(1000);
49 return (int)Skutek_kolizji::SMIERC_WYKONAJACEGO_AKCJE;
50 }
51 else return 0;
52 }
53 }

```

Stworzyłam klasę **Świat**, w której skład wchodzi obiekty klasy **Organizm**. Zaimplementowałam przebieg tury, wywołując metody **akcja()** dla wszystkich organizmów oraz **kolizja()** dla organizmów na tym samym polu. Kolejność wywoływania metody **akcja()** zależy od inicjatywy (lub wieku, w przypadku równych wartości inicjatyw) organizmu. Organizmy mają możliwość wpływania na stan świata. Dlatego istnieje konieczność przekazania metodom **akcja()** oraz **kolizja()** parametru określającego obiekt klasy **Świat**. Postarałam się, aby klasa **Świat** definiowała jako publiczne składowe tylko takie pola i metody, które są potrzebne pozostałym obiektom aplikacji do działania. Pozostałą funkcjonalność świata starałam się zawrzeć w składowych prywatnych.

Z początku miałam problem z rozmnażaniem organizmów. Próbowałam zmienić konstruktor tak, aby przyjmował on referencję do świata, żeby później móc dodać organizm do świata i go rozmnożyć. Na potrzeby rozmnażania/zasiewania organizmów stworzyłam nową klasę **TworzenieOrganizmow**, w której mam metodę, którą wywołuję gdy chcę stworzyć nowy organizm.

Wizualizację świata przeprowadzam w konsoli. Każdy organizm jest reprezentowany przez inny symbol ASCII. Naciśnięcie jednego z klawiszy (Enter) powoduje przejście do kolejnej tury, wyczyszczenie konsoli i ponowne wypisanie odpowiednich symboli, reprezentujących zmieniony stan gry. Co najmniej jedna linia tekstu w konsoli przeznaczona jest na raportowanie wyników zdarzeń takich jak jedzenie lub wynik walki. Zawarłam też instrukcję obsługi klawiszy funkcyjnych na górze w konsoli.

Poniżej wypiszę symbole, których użyłam. Z początku wszystkie organizmy były takiego samego koloru, ale udało mi się w dość estetyczny sposób nadać im charakteru dodając kolory. Teraz jest wesoło ☺. Wielkie litery to zwierzęta, małe to rośliny.

W – wilk

O – owca

L – lis

Z – żółw

A – Antylopa

t – trawa

m – mlecz

g – guarana

w – wilcze jagody

b – barszcz Sosnowskiego

człowiek to biały kwadracik (pozycja: 17,3)

Przebieg gry

Po uruchomieniu gry, pojawia się takie okienko terminala. Instrukcja obsługi znajduje się powyżej.

```

Agnieszka Delmaczynska 184592
Kierowanie czlowiekiem: strzalki < ^
                                v
X - Wyjscie | Enter - kolejna tura | U - wlacz specjalna umiejetnosc

Numer tury: 0
01234567890123456789
0  m w t  O W Z
1  L Z L  ZZ  b
2  O O  g  t
3 w  g Z AO  t
4  L W  m m
5  LA t  t
6  L A W tm g g
7  O LW OL L A
8  L O W mb b
9  L m w  g g
10 w m Z Zg g A
11 L w t W W
12 O A O t W
13 b mw W b
14  ZZA
15 w m O W A W Z
16  bA m  g
17 w O Zb
18 Ab wbm b W
19 gg t t t bA

Liczba organizmow: 111
Nacisnij klawisz

```

```

Nacisnij klawisz
kolizja: Lis x[5] y[10] sila[3] inicjatywa[7], Zolw x[5] y[10] sila[2] inicjatywa[1]
Zolw odgonil atakujacego
Umiera roslina: Trawa
kolizja: Owca x[11] y[1] sila[4] inicjatywa[4], Zolw x[11] y[1] sila[2] inicjatywa[1]
Zolw odgonil atakujacego
Czlowiek nie moze wyjsc z zagrody
kolizja: Owca x[6] y[8] sila[4] inicjatywa[4], Owca x[6] y[8] sila[4] inicjatywa[4]
Rozmnazanie zwierzecia Owca x[6] y[8]
Nowe zwierzecie Owca x[6] y[9]
SZANSA NA UCIECZKE ANTYPLOPY
Antylopa x[7] y[16] Antylopa uniknela kolizji, idzie w gore
Antylopa x[3] y[18] ROBI RUCH GORA 2
Antylopa x[4] y[6] ROBI RUCH PRAWO 2
Antylopa x[4] y[12] ROBI RUCH PRAWO 2
Antylopa x[7] y[15] ROBI RUCH DOL 2
Antylopa x[9] y[5] ROBI RUCH PRAWO 2
Umiera roslina: Trawa
Antylopa x[11] y[14] ROBI RUCH PRAWO 2
Antylopa x[12] y[3] ROBI RUCH LEWO 2
Antylopa x[13] y[15] ROBI RUCH DOL 2
Antylopa x[14] y[7] ROBI RUCH PRAWO 2
Antylopa x[15] y[10] ROBI RUCH PRAWO 2
Antylopa x[15] y[19] ROBI RUCH PRAWO 2
Akcja rosliny: Trawa x[6] y[19] sila[0] inicjatywa[0]
Rozprzestrzenianie rosliny Trawa x[6] y[19] sila[0] inicjatywa[0]
Nowa roslina Trawa x[6] y[18]
Akcja rosliny: Trawa x[7] y[0] sila[0] inicjatywa[0]
Rozprzestrzenianie rosliny Trawa x[7] y[0] sila[0] inicjatywa[0]
Nowa roslina Trawa x[7] y[1]
Akcja rosliny: Trawa x[10] y[12] sila[0] inicjatywa[0]
Rozprzestrzenianie rosliny Trawa x[10] y[12] sila[0] inicjatywa[0]
Nowa roslina Trawa x[10] y[11]

```


Po wykonanej turze:

```

Agnieszka Delmaczyńska 184592
Kierowanie człowiekiem: strzałki < >
X - Wyjście | Enter - kolejna tura | U - włącz specjalna umiejętność
Numer tury: 1
01234567890123456789
0 mm Lw t OZ W Z
1 O O Zt Z bb
2 L gg O tt
3 W L g ZA t
4 W mm m t
5 L m A t
6 WA Lm ggg
7 O L O L A b
8 L W OL Ww mb b
9 L Ommw mb g g
10 W mm Z Zgggg W gA
11 O mww t t
12 L Aw OtW
13 bmmw WW bb
14 b Z Z A
15 wmmm wwm W Z
16 W A bb mmm g
17 W AOwb b b
18 bbt wbwmb b
19 gg t t t bb AW

Liczba organizmów: 146
Nacisnij klawisz

```

Mój main() wygląda tak, nie ma luźnych funkcji niebędących metodami jakiejś klasy (poza mainem):

```

POprojekt1 (Globalny zasięg)
1 #include <iostream>
2 #include <list>
3 #include <Windows.h>
4 #include "Wszystkie_klasy.h"
5 using namespace std;
6
7 int main()
8 {
9     Swiat stworzony_swiat;
10    stworzony_swiat.rysuj_swiat();
11    while (stworzony_swiat.get_czy_jest_koniec_gry() == false)
12    {
13        stworzony_swiat.wykonaj_ture();
14        stworzony_swiat.rysuj_swiat();
15        Sleep(200);
16    }
17 }

```

Zrealizowałam funkcjonalności wymagane do uzyskania 4 punktów. Poniżej wstawiam skopiowaną punktację z instrukcji, żeby nie pominąć czegoś, co zrealizowałam.

Punktacja:

- 3 punkty

- o Implementacja świata gry i jego wizualizacji.

- o Implementacja wszystkich obowiązkowych gatunków zwierząt, bez rozmnażania.

- o Implementacja wszystkich gatunków roślin, bez rozprzestrzeniania.

- o Implementacja Człowieka poruszanego za pomocą strzałek na klawiaturze.

- 4 punkty Jak wyżej oraz dodatkowo o rozmnażanie się zwierząt i rozprzestrzenianie się roślin, o oraz implementacja specjalnej umiejętności Człowieka.

Po skończeniu tego projektu, czuję się już pewniej w pisaniu kodu obiektowo.

Nadal pozostało sporo materiału do nauki, ale myślę, że praca czyni mistrza i stopniowo oraz systematycznie pracując pogłębiamy swoje umiejętności.