**Programowanie obiektowe – raport projekt 1 C++**

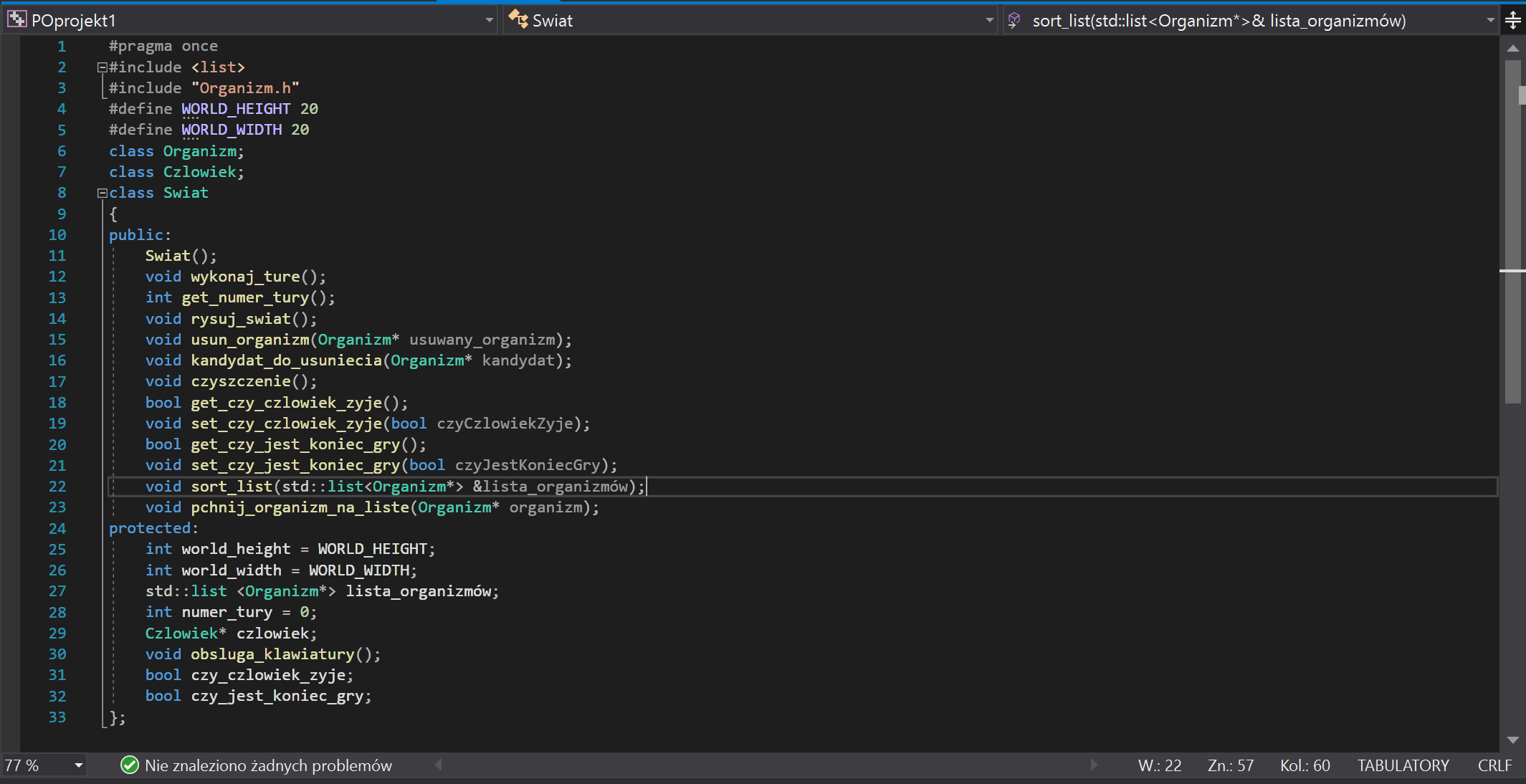
W poniższym dokumencie opiszę pracę, którą wykonałam podczas projektu pt. **„Wirtualny świat”** w ramach przedmiotu Programowanie obiektowe.

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**;”.



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**:

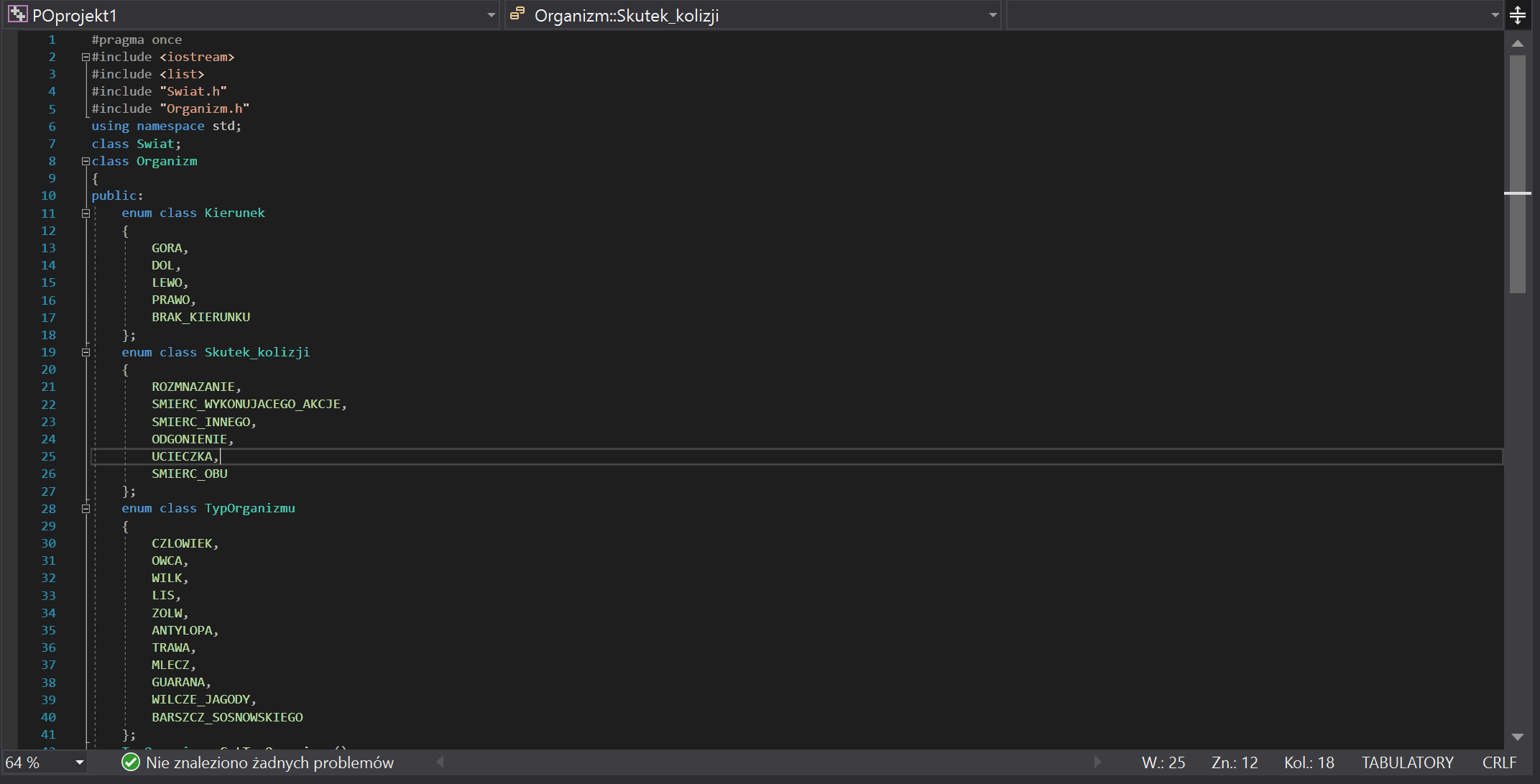
• **akcja()** → określa zachowanie organizmu w trakcie tury,

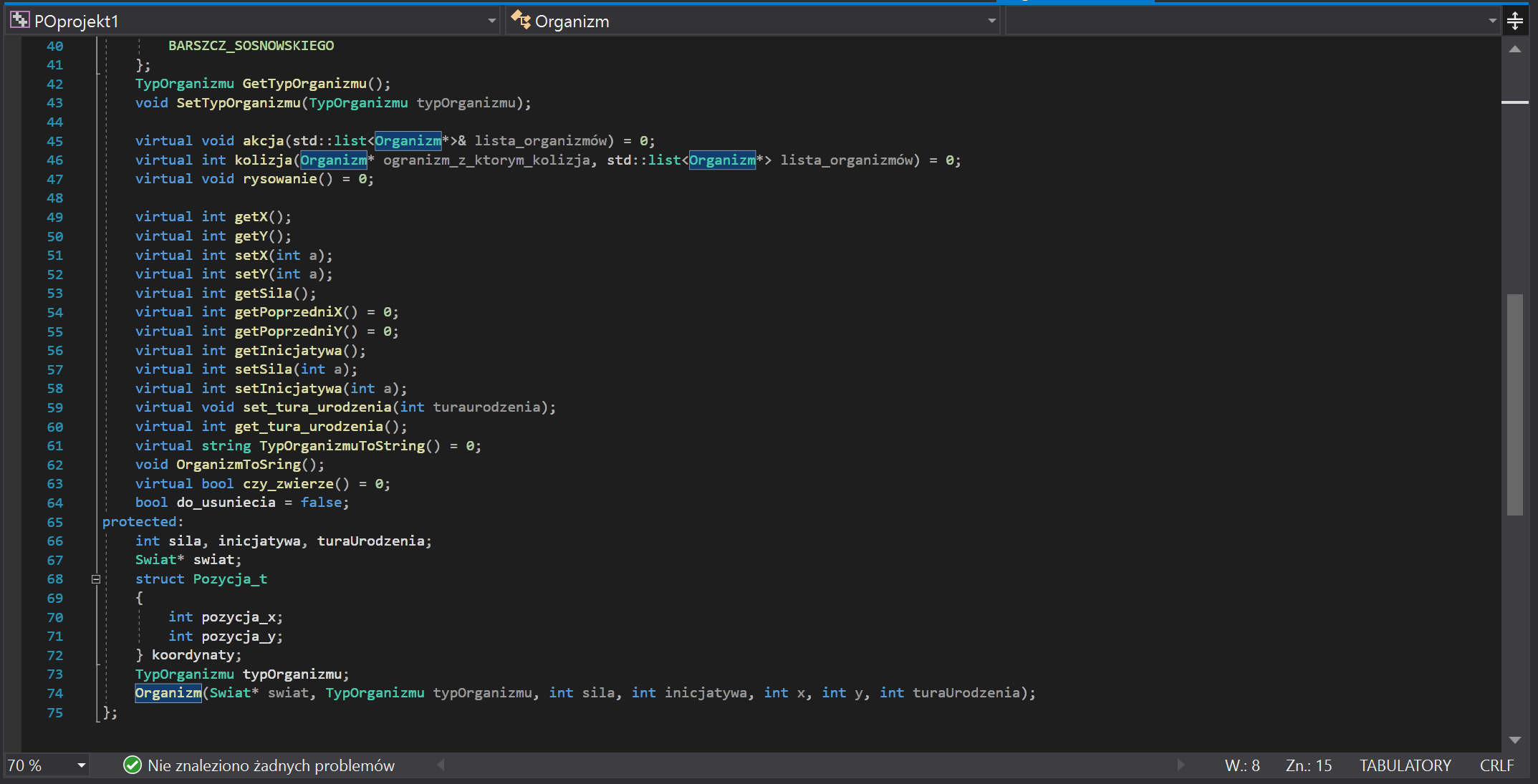
• **kolizja()** → określa zachowanie organizmu w trakcie kontaktu/zderzenia z innym organizmem,

• **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**:

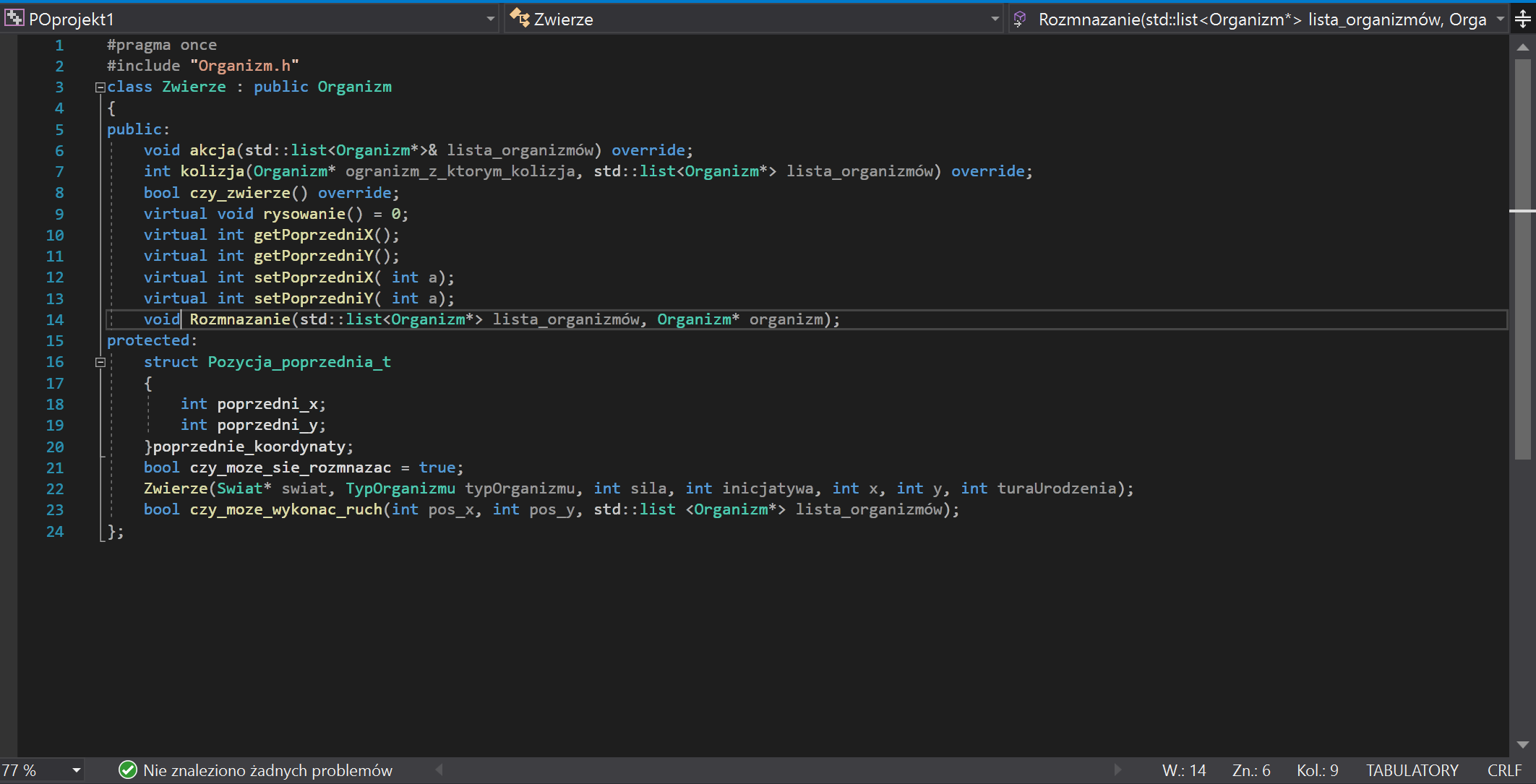


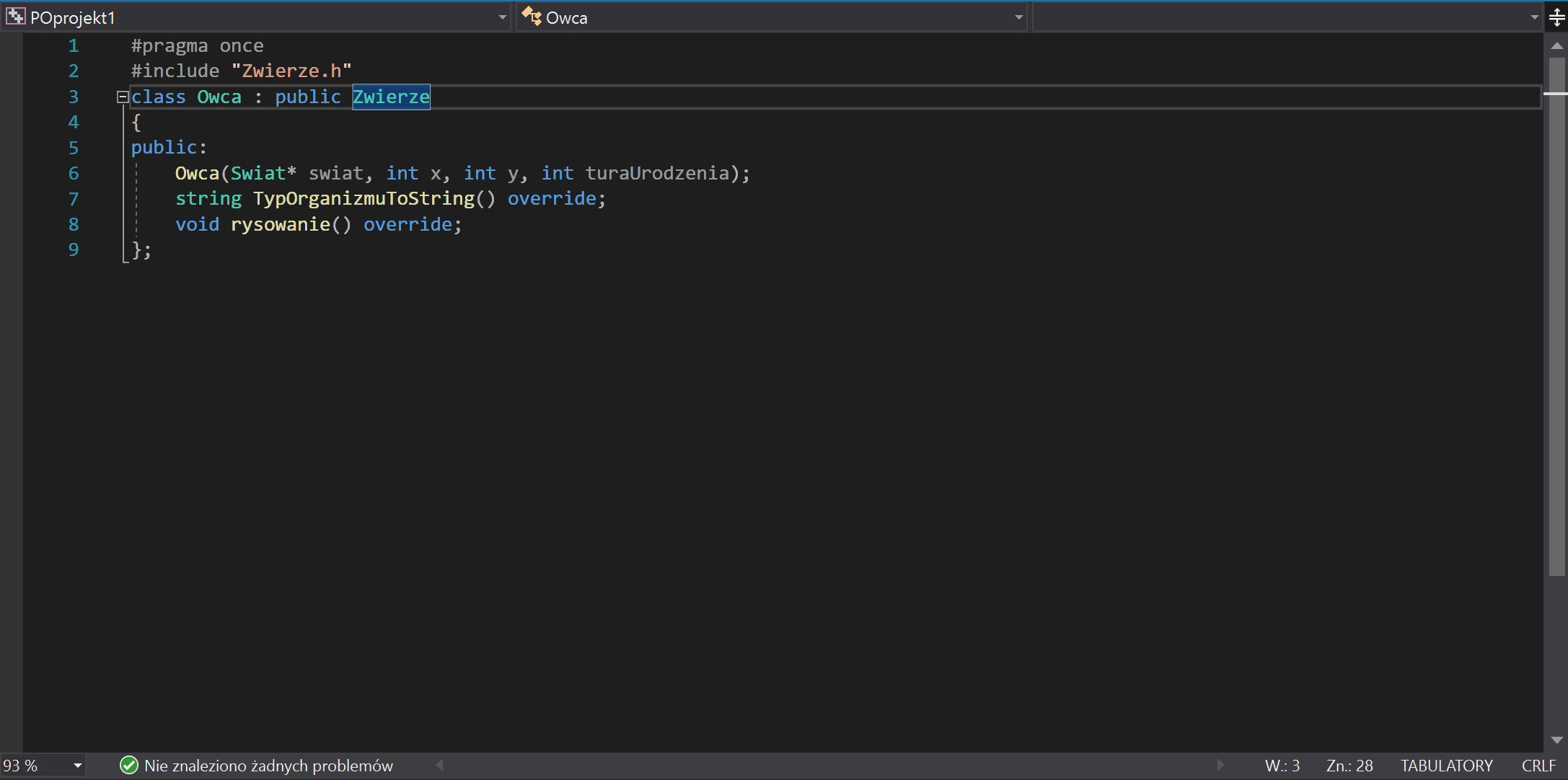
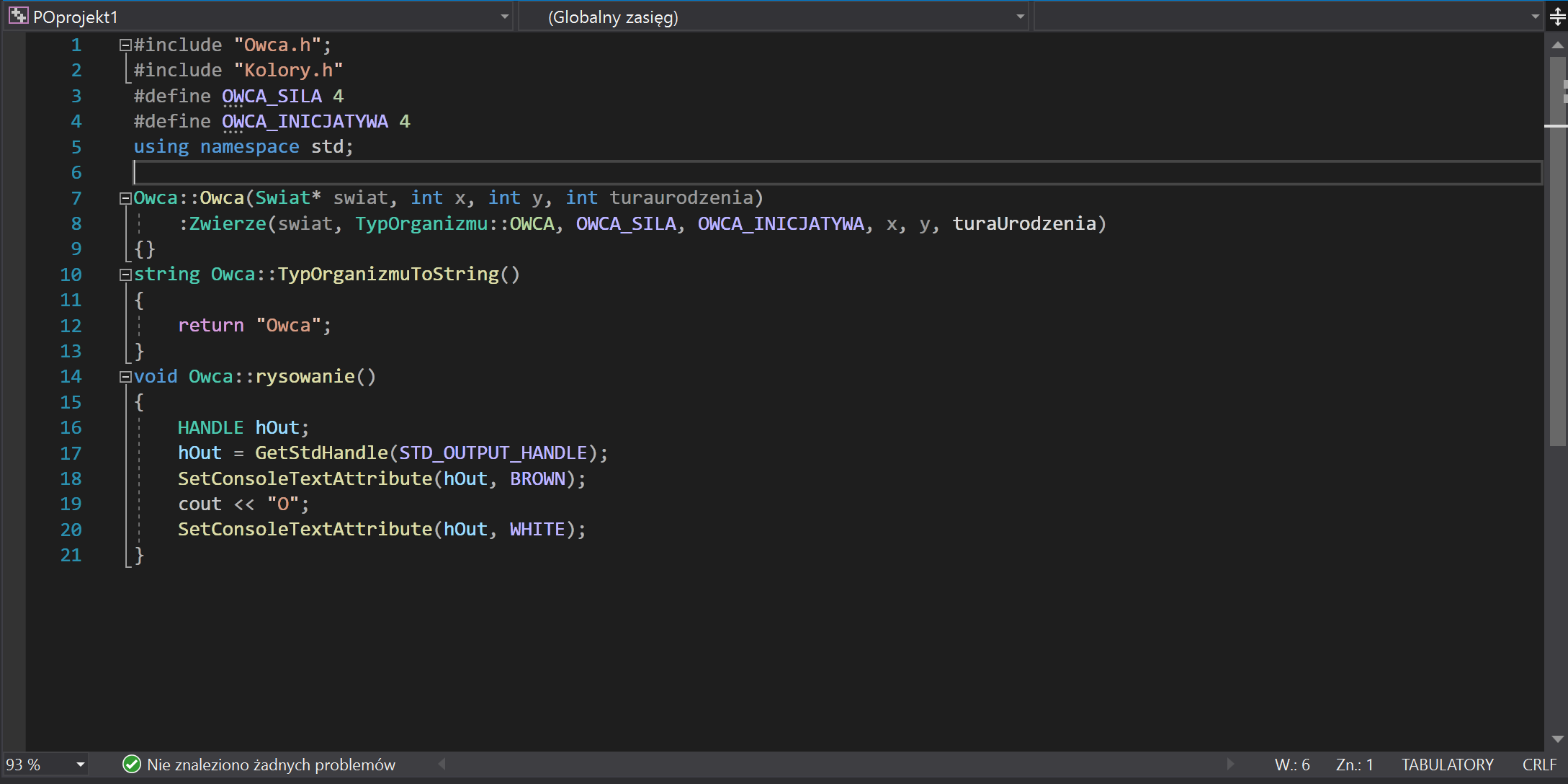


W klasie **Zwierze** zaimplementowałam wspólne dla wszystkich/większości zwierząt zachowania, przede wszystkim:

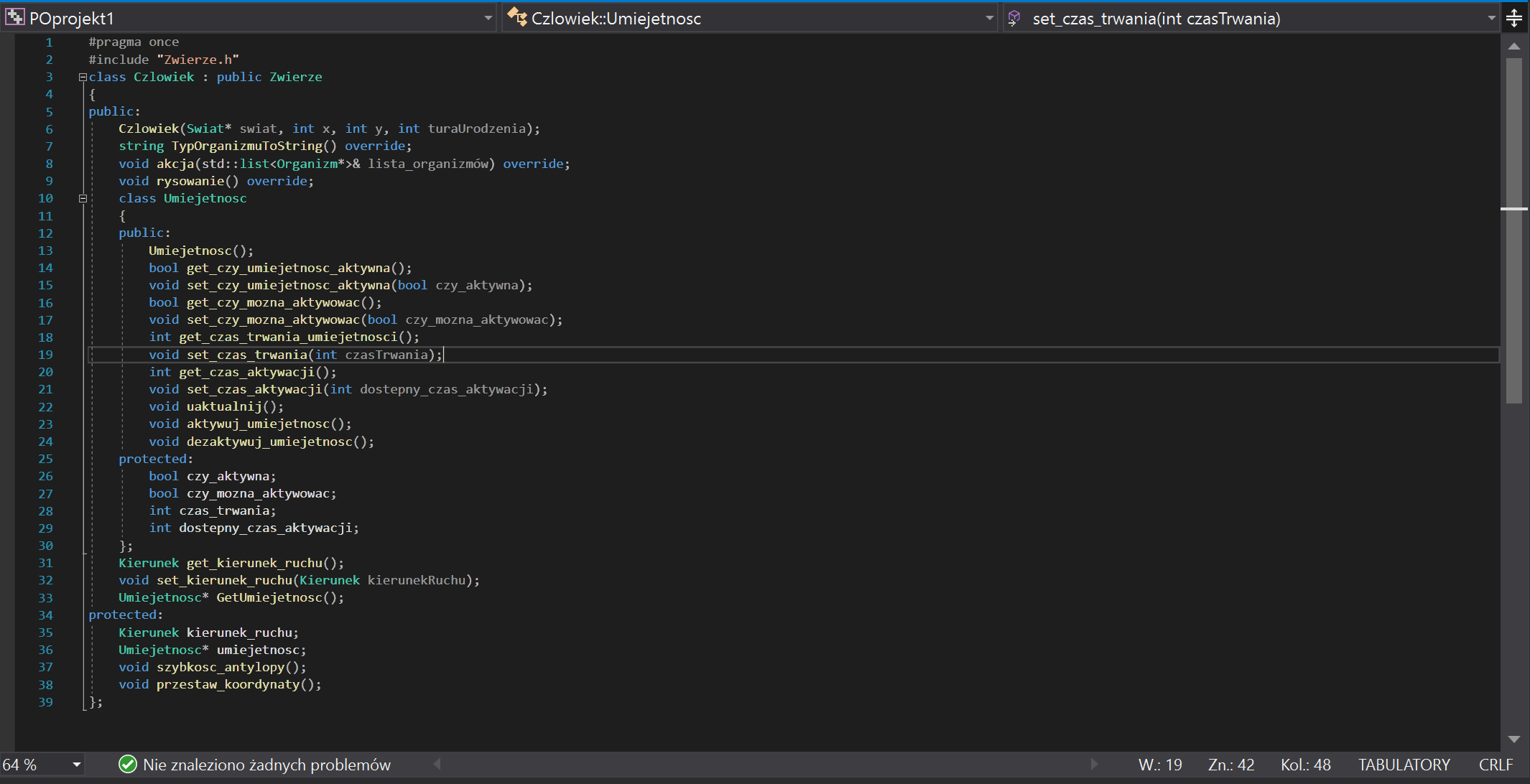
• podstawową formę ruchu w metodzie **akcja()** → każde typowe zwierzę w swojej turze przesuwa się na wybrane losowo, sąsiednie pole,

• 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.



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

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 **obsluga\_klawiatury()** (**protected w Swiat.h**) i stworzyłam plik nagłówkowy **Klawiatura.h** przechowujący key symbols klawiszy funkcyjnych.



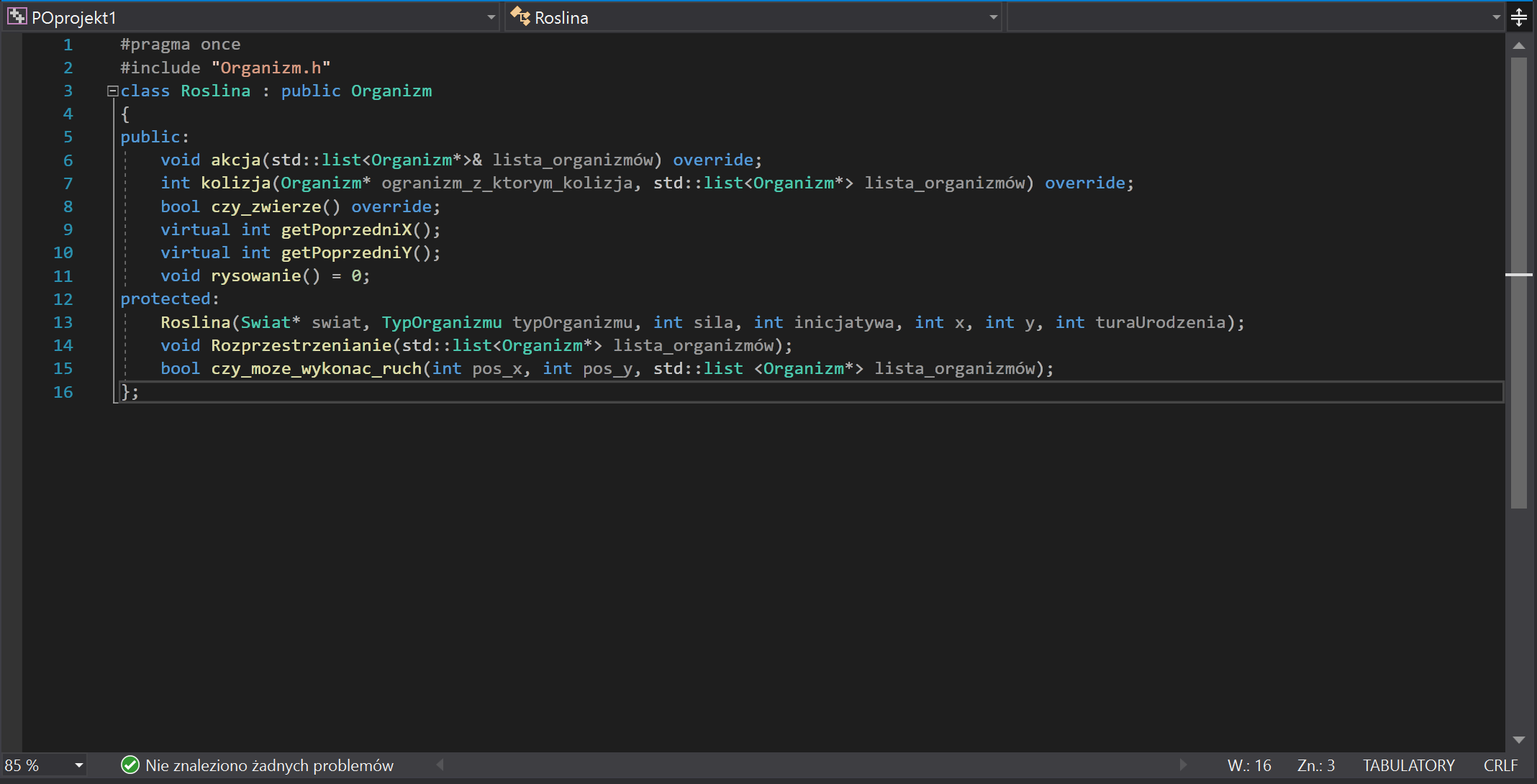
Udało mi się zrealizować projekt na wymagania na 4 pkt, więc projekt nie posiada klawisza funkcyjnego Save. Projekt posiada **klawisze funkcyjne**: **X – wyjście z gry, U – aktywacja specjalnej umiejętności człowieka i Enter, czyli przejście do kolejnej tury**, co jest jednoznaczne z utratą ruchu człowieka.

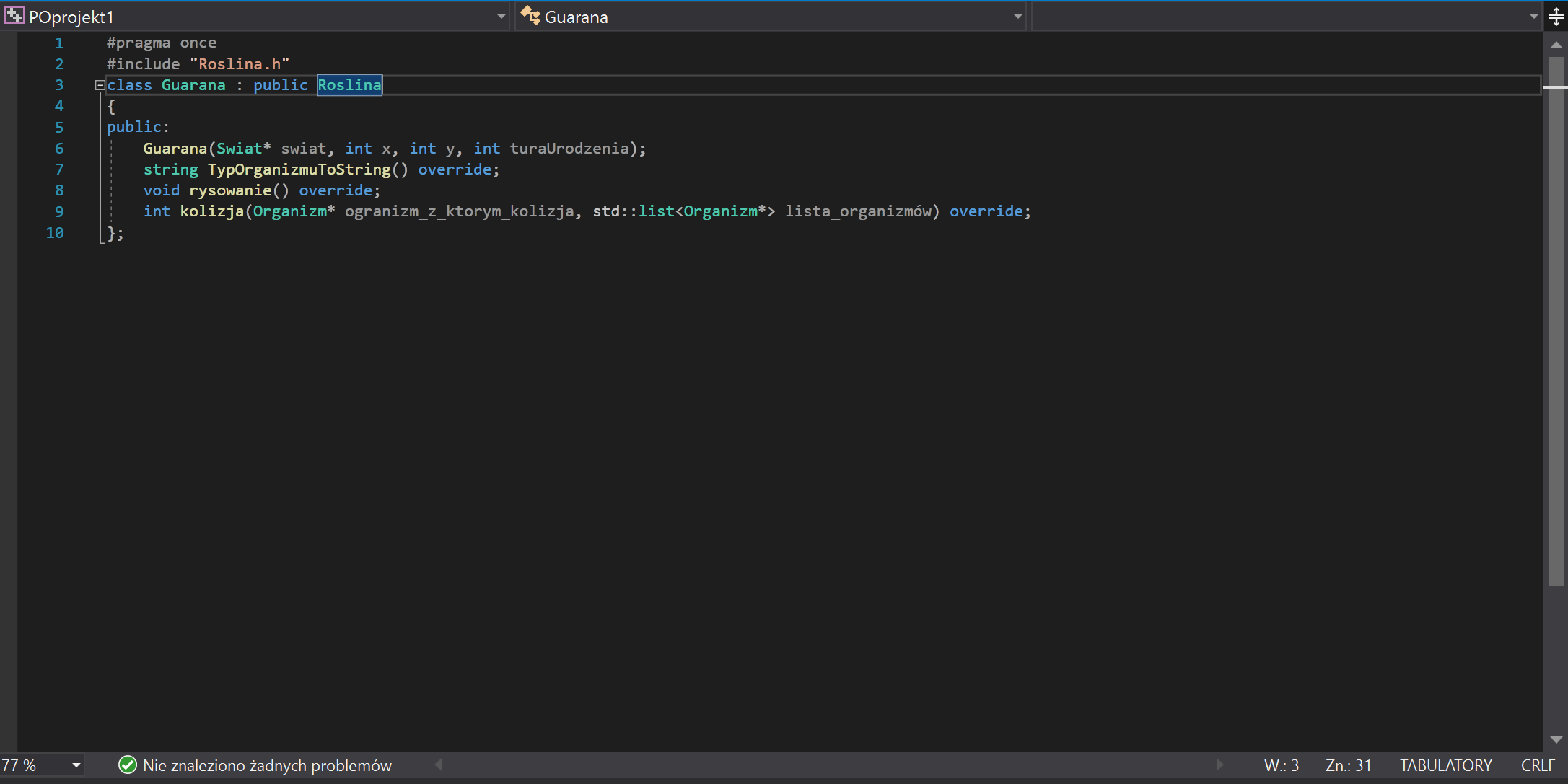
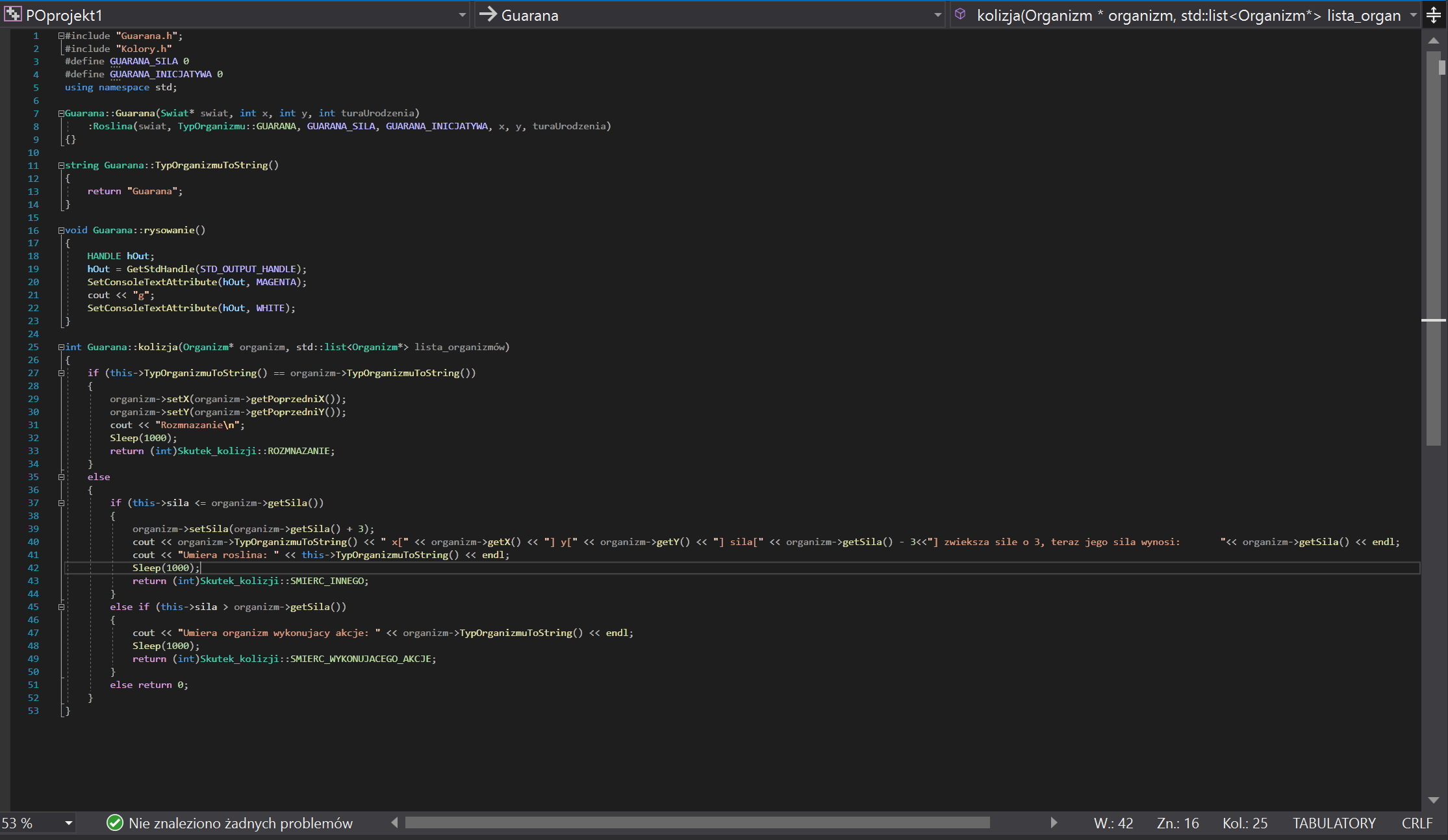
Człowiek jest tylko jeden i się nie rozmnaża. Jego specjalną umiejętnością jest szybkość antylopy (mój indeks: 184592, (ostatnia cyfra indeksu)mod 5 == 2, 2 to szybkość antylopy). Po włączeniu tej umiejętności klikając U, człowiek zyskuje umiejętność. Jej czas trwania wynosi 5 tur, a możliwość ponownej 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, mlecz, guarana, wilcze jagody i barszcz Sosnowskiego.



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

Stworzyłam klasę **Świat**, w której skład wchodzą 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**

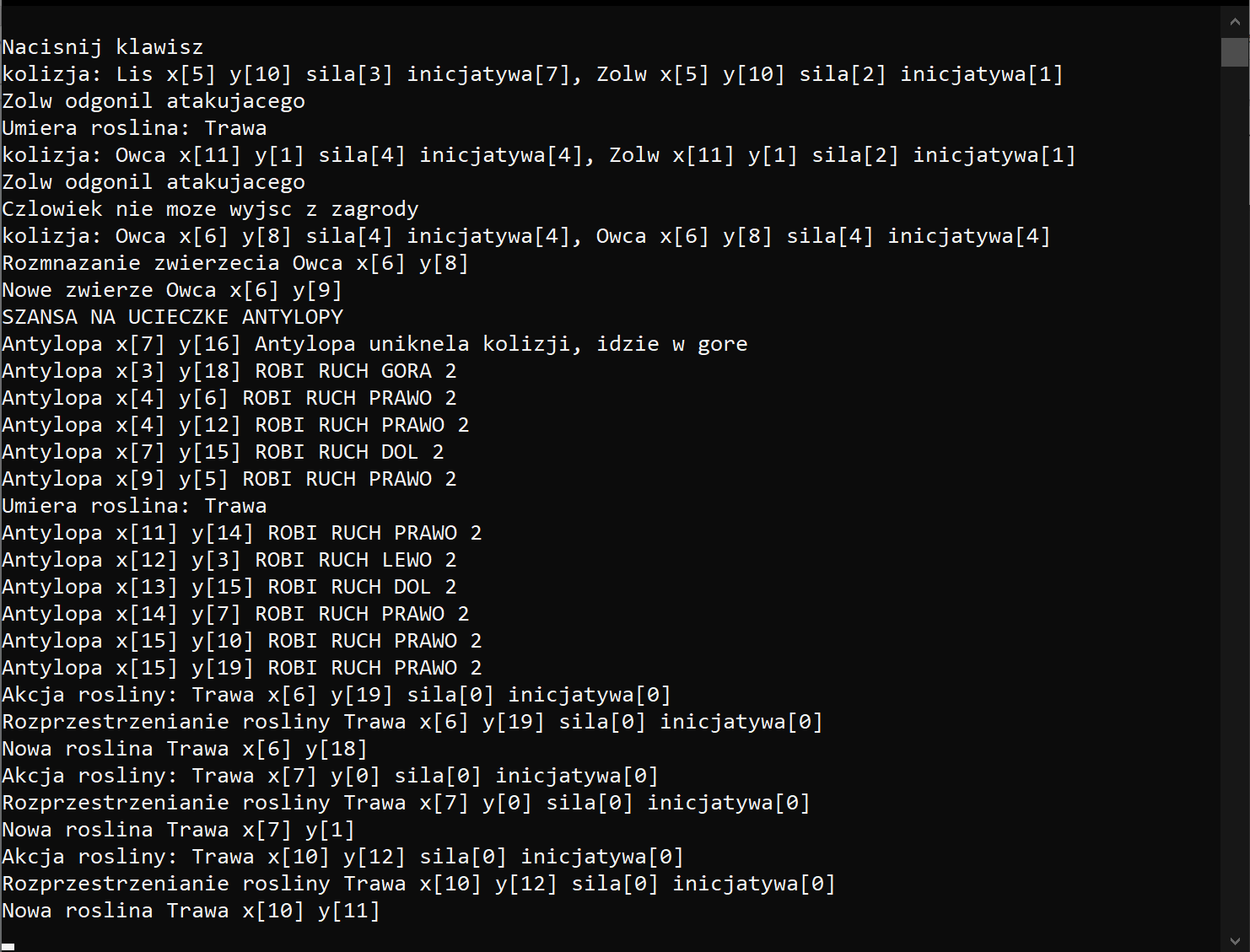
**b – barszcz Sosnowskiego**

**człowiek to biały kwadracik** (pozycja: 17,3)

**Przebieg gry**

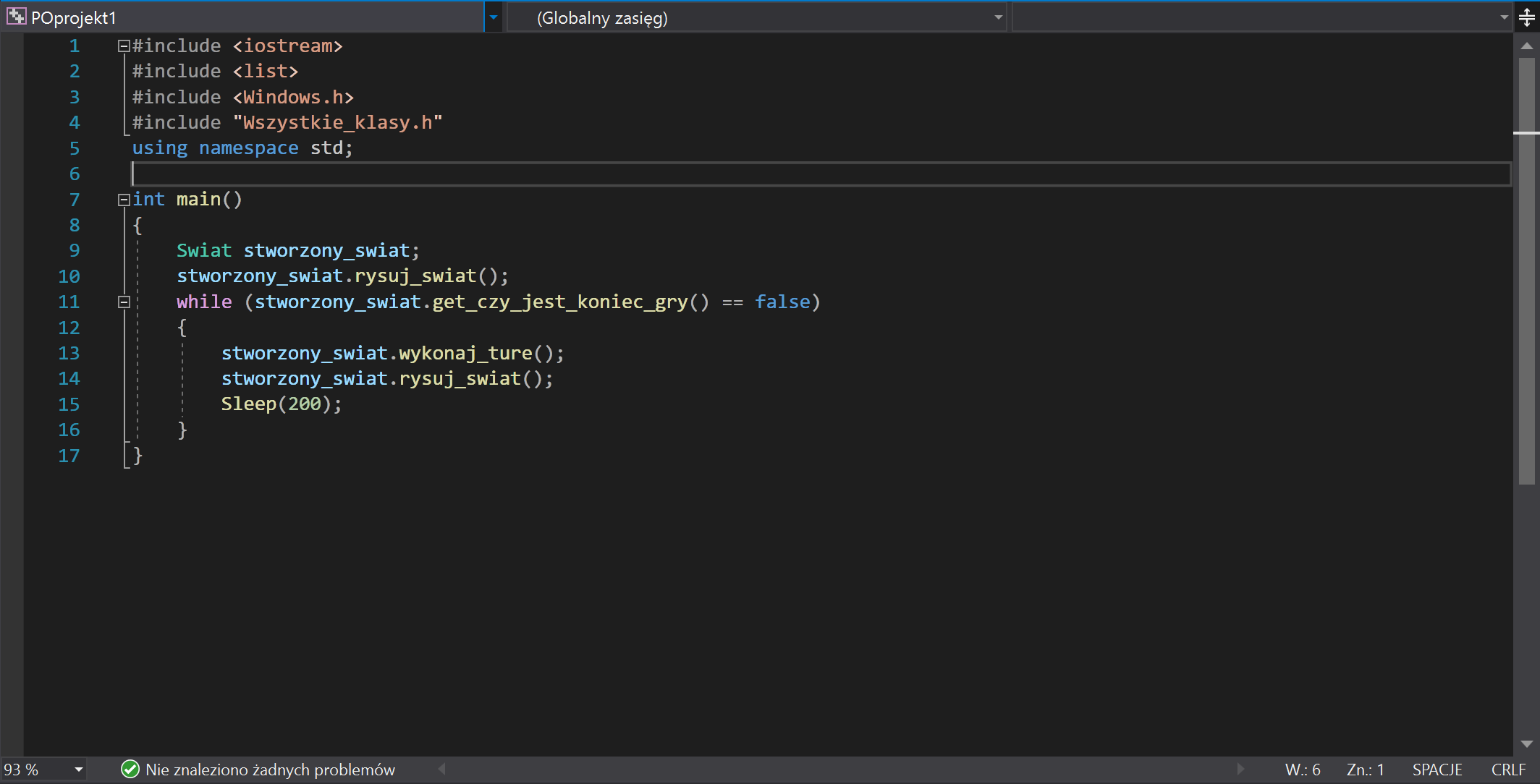
Po uruchomieniu gry, pojawia się takie okienko terminala. Instrukcja obsługi znajduje się powyżej.





**Po wykonanej turze:**



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

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.