

# Programowanie Obiektowe



## Laboratorium otwarte – opis zadań



Celem projektu jest implementacja programu o charakterze symulatora wirtualnego świata, który ma mieć strukturę dwuwymiarowej kraty o dowolnym, zadanym przez użytkownika rozmiarze NxM (na 4 punkty można poprzestać na stałym rozmiarze 20x20). W świecie tym będą istniały proste formy życia o odmiennym zachowaniu. Każdy z organizmów zajmuje dokładnie jedno pole w tablicy, na każdym polu może znajdować się co najwyżej jeden organizm (w przypadku kolizji jeden z nich powinien zostać usunięty lub przesunięty).

Symulator ma mieć charakter turowy. W każdej turze wszystkie organizmy istniejące na świecie mają wykonać akcję stosowną do ich rodzaju. Część z nich będzie się poruszała (organizmy zwierzęce), część będzie nieruchoma (organizmy roślinne). W przypadku kolizji (jeden z organizmów znajdzie się na tym samym polu, co inny) jeden z organizmów zwycięża, zabijając (np. wilk) lub odganiając (np. żółw) konkurenta. Kolejność ruchów organizmów w turze zależy od ich inicjatywy. Pierwsze ruszają się zwierzęta posiadające najwyższą inicjatywę. W przypadku zwierząt o takiej samej inicjatywie o kolejności decyduje zasada starszeństwa (pierwszy rusza się dłużej żyjący). Zwycięstwo przy spotkaniu zależy od siły organizmu, choć będą od tej zasady wyjątki (patrz: Tabela 2). Przy równej sile zwycięża organizm, który zaatakował. Specyficznym rodzajem zwierzęcia ma być Człowiek. W przeciwieństwie do zwierząt, człowiek nie porusza się w sposób losowy. Kierunek jego ruchu jest determinowany przed rozpoczęciem tury za pomocą klawiszy strzałek na klawiaturze. Człowiek posiada też specjalną umiejętność (patrz Załącznik 1) którą można aktywować osobnym przyciskiem. Aktywowana umiejętność pozostaje czynna przez 5 kolejnych tur, po czym następuje jej dezaktywacja. Po dezaktywacji umiejętność nie może być aktywowana przed upływem 5 kolejnych tur. Przy uruchomieniu programu na planszy powinno się pojawić po kilka sztuk wszystkich rodzajów zwierząt oraz roślin. Okno programu powinno zawierać pole, w którym wypisywane będą informacje o rezultatach walk, spożyciu roślin i innych zdarzeniach zachodzących w świecie.

**W interfejsie aplikacji musi być przedstawione: imię, nazwisko oraz numer indeksu autora.**

Poniższe uwagi nie obejmują wszystkich szczegółów i są jedynie wskazówkami do realizacji projektu zgodnie z regułami programowania obiektowego.

Należy utworzyć klasę **Świat** (Swiat) zarządzającą rozgrywką i organizmami. Powinna zawierać m.in. metody, np:

- wykonajTure()
- rysujSwiat()

oraz pola, np.:

- organizmy

Należy również utworzyć abstrakcyjną klasę **Organizm**.

podstawowe pola:

- siła
- inicjatywa
- położenie (x,y).
- świat - referencja do świata w którym znajduje się organizm

podstawowe metody:

- akcja() → określa zachowanie organizmu w trakcie tury,
- kolizja() → określa zachowanie organizmu w trakcie kontaktu/zderzenia z innym organizmem,
- rysowanie() → powoduje narysowanie symbolicznej reprezentacji organizmu.

Klasa **Organizm** powinna być abstrakcyjna. Dziedziczyć po niej powinny dwie kolejne abstrakcyjne klasy: **Roślina** oraz **Zwierzę**.

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

- podstawową formę ruchu w metodzie `akcja()` → każde typowe zwierze 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 zwierze, tego samego gatunku.

Klasa **Człowiek** ma stanowić rozszerzenie klasy **Zwierzę**. Nie posiada on własnej inteligencji (sterowany jest przez gracza) oraz nie rozmnaża się (gracz będzie jedynym Człowiekiem na mapie).

*Tabela 1. Charakterystyka klasy Człowiek.*

siła	inicjatywa	specyfika metody <code>akcja()</code>	specyfika metody <code>kolizja()</code>
5	4	Człowiek porusza się w taki sam sposób jak zwierzęta, ale kierunek jego ruchu nie jest przypadkowy, a odpowiada naciśniętej przez gracza strzałce na klawiaturze. Tzn. jeżeli gracz naciśnie strzałkę w lewo, to (gdy nadejdzie jej kolej) postać przesunie się o jedno pole w lewo.	Człowiek posiada specjalną umiejętność (patrz załącznik 1), którą można aktywować osobnym przyciskiem na klawiaturze. Po aktywowaniu umiejętność ta wpływa na zachowanie metody <code>kolizja()</code> przez pięć kolejnych tur. Następnie umiejętność zostaje wyłączona i nie może być ponownie aktywowana przez pięć następnych tur.

Zaimplementuj 5-6 klas zwierząt. Rodzaje zwierząt definiuje poniższa tabela.

*Tabela 2. Spis zwierząt występujących w wirtualnym świecie.*

Id	zwierzę	siła	inicjatywa	specyfika metody <code>akcja()</code>	specyfika metody <code>kolizja()</code>
1	wilk	9	5	brak	brak
2	owca	4	4	brak	brak
3	lis	3	7	Dobry wąż: lis nigdy nie ruszy się na pole zajmowane przez organizm silniejszy niż on	brak
4	żółw	2	1	W 75% przypadków nie zmienia swojego położenia.	Odpiera ataki zwierząt o sile <5. <b>Napastnik musi wrócić na swoje poprzednie pole.</b>
5	antylopa	4	4	Zasięg ruchu wynosi 2 pola.	50% szans na ucieczkę przed walką. Wówczas przesuwa się na niezajęte sąsiednie pole.

<i>Id</i>	<i>zwierzę</i>	<i>sila</i>	<i>inicjatywa</i>	<i>specyfika metody akcja()</i>	<i>specyfika metody kolizja()</i>
6	cyber-owca <sup>1</sup>	11	4	Jej celem nadrzędnym jest eksterminacja barszczu sosnowskiego. Zawsze kieruje się w stronę najbliższego barszczu i próbuje go zjeść. Jeśli na planszy nie ma żadnego barszczu to udaje zwyczajną owcę	Zjada barszcz sosnowskiego.

W klasie **Roślina** zaimplementuj 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.

Zaimplementuj 5 klas roślin. Rodzaje roślin definiuje poniższa tabela.

*Tabela 3. Spis roślin występujących w wirtualnym świecie.*

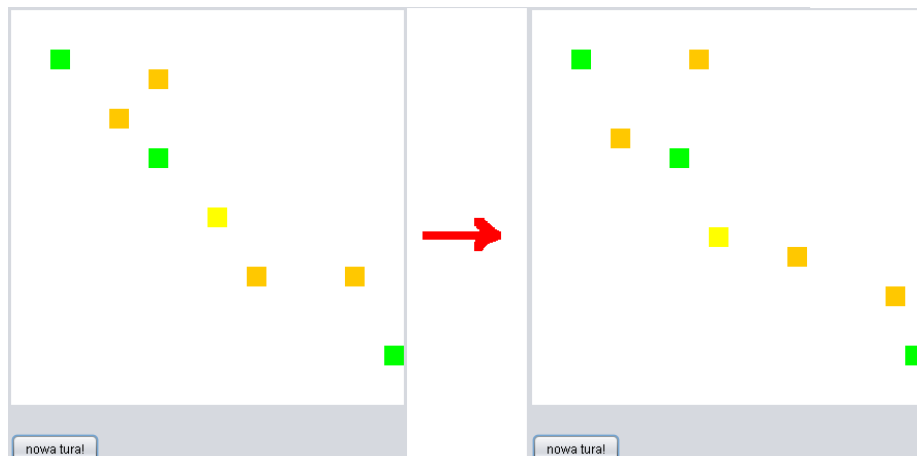
<i>Id</i>	<i>roślina</i>	<i>sila</i>	<i>specyfika metody akcja()</i>	<i>specyfika metody kolizja()</i>
1	trawa	0	brak	brak
2	mlecz	0	Podejmuje trzy próby rozprzestrzeniania w jednej turze	brak
3	guarana	0	brak	Zwiększa siłę zwierzęcia, które zjadło tę roślinę, o 3.
4	wilcze jagody	99	brak	Zwierze, które zjadło tę roślinę, ginie.
5	barszcz sosnowskiego	10	Zabija wszystkie zwierzęta w swoim sąsiedztwie poza cyber-owcą.	Zwierze które zjadło tę roślinę ginie. Jedynie cyber-owca jest odporna.

Stwórz klasę **Świat** w której skład wchodzi obiekty klasy **Organizm**. Zaimplementuj przebieg tury, wywołując metody `akcja()` dla wszystkich organizmów oraz `kolizja()` dla organizmów na tym samym polu. Pamiętaj, że 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**. Postaraj się, aby klasa **Świat** definiowała jako publiczne składowe tylko takie pola i metody, które są potrzebne pozostałym obiektom aplikacji do działania. Pozostałą funkcjonalność świata staraj się zawrzeć w składowych prywatnych.

<sup>1</sup> Cyber-owca jest obowiązkowa jedynie w projekcie nr 3 w języku Python

Przykładowy wygląd aplikacji, którą należy zaimplementować (w wariacie graficznym) przedstawia poniższa para rysunków.



***Rysunek 1. Ilustracja zasady działania świata wirtualnego.***

### **Projekt 1. C++**

Wizualizację świata należy przeprowadzić w konsoli. Każdy organizm jest reprezentowany przez inny symbol ASCII. Naciśnięcie jednego z klawiszy 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.

Punktacja:

- 3 punkty
  - Implementacja świata gry i jego wizualizacji.
  - Implementacja wszystkich obowiązkowych gatunków zwierząt, bez rozmnażania.
  - Implementacja wszystkich gatunków roślin, bez rozprzestrzeniania.
  - Implementacja Człowieka poruszanego za pomocą strzałek na klawiaturze.
- 4 punkty Jak wyżej oraz dodatkowo
  - rozmnażanie się zwierząt i rozprzestrzenianie się roślin,
  - oraz implementacja specjalnej umiejętności Człowieka.
- 5 punktów
  - Implementacja możliwości zapisania do pliku i wczytania z pliku stanu wirtualnego świata.

**Proszę wystrzegać się błędów wymienionych w załączniku 2. Zaleca się stosowanie dobrych praktyk wymienionych w załączniku 3.**

## Projekt 2. Java

Stwórz aplikację analogiczną jak w języku C++. Tym razem wymagane jest użycie reprezentacji graficznej z wykorzystaniem biblioteki Swing. Funkcje aplikacji (takie jak przejście do kolejnej tury czy zapis i wczytanie stanu świata) realizuj przez komponenty biblioteki Swing, takie jak przyciski i elementy menu.

Punktacja:

- 3 punkty
  - Implementacja świata gry i jego wizualizacji.
  - Implementacja wszystkich obowiązkowych gatunków zwierząt.
  - Implementacja wszystkich gatunków roślin.
  - Implementacja Człowieka poruszanego za pomocą strzałek na klawiaturze.
  - Implementacja specjalnej umiejętności Człowieka.
  - Implementacja możliwości zapisania do pliku i wczytania z pliku stanu wirtualnego świata.
- 4 punkty
  - Implementacja możliwości dodawania organizmów do świata gry. Naciśnięcie na wolne pole powinno dać możliwość dodania każdego z istniejących w świecie organizmów.
- 5 punktów
  - Implementacja abstrakcji dla świata wraz z dwiema implementacjami. W jednej implementacji rozgrywka na kracie, w drugiej implementacji rozgrywka na planszy podzielonej na pola sześciennie (jak w grze planszowej hex).

**Proszę wystrzegać się błędów wymienionych w załączniku 2. Zaleca się stosowanie dobrych praktyk wymienionych w załączniku 3.**

## Projekt 3. Python

Zaimplementuj aplikację analogiczną jak w poprzednim zadaniu, z użyciem języka Python i dowolnej biblioteki graficznej (5pkt., punktacja jak w Projekcie 2).

- 3 punkty
  - Implementacja świata gry i jego wizualizacji.
  - Implementacja wszystkich obowiązkowych gatunków zwierząt (wraz z cyber-owcą).
  - Implementacja wszystkich gatunków roślin.
  - Implementacja Człowieka poruszanego za pomocą strzałek na klawiaturze.
  - Implementacja specjalnej umiejętności Człowieka.
  - Implementacja możliwości zapisania do pliku i wczytania z pliku stanu wirtualnego świata.
- 4 punkty
  - Implementacja możliwości dodawania organizmów do świata gry. Naciśnięcie na wolne pole powinno dać możliwość dodania każdego z istniejących w świecie organizmów.
- 5 punktów
  - Implementacja abstrakcji dla świata wraz z dwiema implementacjami. W jednej implementacji rozgrywka na kracie, w drugiej implementacji rozgrywka na planszy podzielonej na pola sześciennie (jak w grze planszowej hex).

**Proszę wystrzegać się błędów wymienionych w załączniku 2. Zaleca się stosowanie dobrych praktyk wymienionych w załączniku 3.**

**Załącznik 1. Sposób przydziału specjalnej umiejętności Człowieka poszczególnym studentom.**

Przydział specjalnej umiejętności do implementacji jest zdeterminowany numerem indeksu oraz inicjałami autora.

Przydział jest realizowany w następujący sposób:

$$ID = X \bmod 5$$

gdzie:

$ID$  – id (wg poniższej tabeli) specjalnej umiejętności Człowieka

$X$  – ostatnia cyfra numeru indeksu

**Tabela 4. Specjalne umiejętności Człowieka.**

<b>Id</b>	<b>Umiejętność</b>	<b>Cechy</b>
0	Nieśmiertelność	Człowiek nie może zostać zabity. W przypadku konfrontacji z silniejszym przeciwnikiem przesuwa się na losowo wybrane pole sąsiadujące.
1	Magiczny Elixir	Siła Człowieka rośnie do 10 w pierwszej turze działania umiejętności. W każdej kolejnej turze maleje o „1”, aż wróci do stanu początkowego.
2	Szybkość antylopy	Człowiek w porusza się na odległość dwóch pól zamiast jednego przez pierwsze 3 tury działania umiejętności. W pozostałych 2 turach szansa że umiejętność zadziała ma wynosić 50%.
3	Tarcza Alzura	Człowiek odstrasza wszystkie zwierzęta. Zwierzę które stanie na polu Człowieka zostaje przesunięte na losowe pole sąsiednie.
4	Całopalenie	Człowiek niszczy wszystkie rośliny i zwierzęta znajdujące się na polach sąsiadujących z jego pozycją.

## Załącznik 2. Przykładowe błędy za które będzie obniżana ocena

- Brak polimorfizmu i przynajmniej jednej klasy abstrakcyjnej (-1 pkt)
- Nadużywanie funkcji statycznych/luźnych funkcji (-1 pkt)

W projekcie C++ jedyną luźną funkcją może być `main`. Proszę nie nadużywać słowa kluczowego ***static*** i używać go tylko tam, gdzie jest to konieczne.

- Brak hermetyzacji (-1 pkt.)

Pola klas powinny być chronione modyfikatorami ***private*** lub ***protected***. Pola o zakresie widoczności ***public*** mogą być stosowane tylko wtedy, gdy reprezentują wartości stałe (***const/final***), najlepiej, aby były statyczne.

- Cechy specjalne organizmów zaimplementowane poza klasą, dla której są właściwe (-1 pkt)

Przykładowo błędem będzie zaimplementowanie w metodzie `kolizja()` klasy `Zwierze` sprawdzania czy organizm z którym zderzył się organizm atakujący jest klasy `Zolw`, a następnie porównywanie siły i pozostawanie na tym samym polu jeśli jest to konieczne. Lepiej dodać metodę wirtualną `bool czyOdbilAtak(Organizm& atakujacy)` do organizmu lub zwierzęcia i następnie wywołać ją w trakcie kolizji. Dzięki temu można także zaimplementować tarczę Alzura.



## Załącznik 3. Sugerowane dobre praktyki programistyczne

### Klasy i obiekty

- Dobrze jest stosować logiczny podział na przestrzenie nazw - każda przestrzeń nazw w oddzielnym module (pliku).
- Metody które nie wykorzystują obiektu powinny być statyczne. Nie należy ich nadużywać.

### Hermetyzacja

Dobrze jest aby wybrane klasy miały metody typu get i set dla składowych lub tylko get, lub całkowity brak dostępu bezpośredniego.

### Dziedziczenie

Wielokrotne wykorzystanie kodu (kod w klasie bazowej używany przez obiekty klas pochodnych). Podczas nadpisywania metody klasy bazowej można dalej wykorzystywać implementacje z klasy nadrzędnej (bazowej).

### Polimorfizm

- Warto wykorzystać metody służące do dynamicznego określania typu:
  - `dynamic_cast<klasa_pochodna*>()` dla C++
  - `instanceof` dla Javy
  - `isinstance()` lub `type(a) is ...` dla Pythona
- Warto stosować interfejsy oraz klasy abstrakcyjne.

### Inne

Warto zastosować wyjątki do sygnalizacji i obsługi błędów.

- <http://en.cppreference.com/w/cpp/language/throw> dla C++
- <https://docs.oracle.com/javase/tutorial/essential/exceptions/index.html> dla Javy
- <https://docs.python.org/2/tutorial/errors.html> dla Pythona

### Styl programowania

Dobrze jest przestrzegać reguł związanych ze stylem programowania. Można w tym celu wykorzystać zasady zamieszczone w artykule:

<http://geosoft.no/development/cppstyle.html>

przede wszystkim ważne są:

- spójność nazewnictwa zmiennych i typów,
- spójność w zakresie stosowania tabulacji (wcięcia) i odstępów,
- ograniczony rozmiar funkcji,
- zachowanie spójności w organizacji kodu źródłowego wewnątrz klasy (np. jednolita kolejność `public->protected->private`).