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 akcja()	specyfika metody kolizja()
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 kolizja() przez pięć kolejnych tur. Następnie umiejętność zostaje wyłączona i nie może być ponownie aktywowana przez pięć następnych tur.

Zaimplementuj 5 klas zwierząt. Rodzaje zwierząt definiuje poniższa tabela.

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

Id	zwierzę	siła	inicjatywa	specyfika metody akcja()	specyfika metody kolizja()
1	wilk	9	5	brak	brak
2	owca	4	4	brak	brak
3	lis	3	7	Dobry węch: 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. **Napastnik musi wrócić na swoje poprzednie pole.**
5	antylopa	4	4	Zasięg ruchu wynosi 2 pola.	50% szans na ucieczkę przed walką. Wówczas przesuwa się na niezajęte sąsiednie pole.

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 4 klasy roślin. Rodzaje roślin definiuje poniższa tabela.

Tabela 3. Spis roślin występujących w wirtualnym świecie.

roślina	siła	specyfika metody akcja()	specyfika metody kolizja()
trawa	0	brak	brak
mlecz	0	Podejmuje trzy próby rozprzestrzeniania w jednej turze	brak
guarana	0	brak	Zwiększa siłę zwierzęcia, które zjadło tę roślinę, o 3.
wilcze jagody	99	brak	Zwierze, które zjadło tę roślinę, ginie.

Stwórz klasę **Świat** w której skład wchodzą 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.

Przykładowy wygląd aplikacji, którą należy zaimplementować (w wariancie 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:
3pkt. Implementacja świata gry i jego wizualizacji. Implementacja wszystkich gatunków zwierząt, bez rozmnażania. Implementacja wszystkich gatunków roślin, bez rozprzestrzeniania. Implementacja Człowieka poruszanego za pomocą strzałek na klawiaturze.
4 pkt. Jak wyżej + rozmnażanie się zwierząt i rozprzestrzenianie się roślin, oraz implementacja specjalnej umiejętności Człowieka.
5 pkt. Implementacja możliwości zapisania do pliku i wczytania z pliku stanu wirtualnego świata.
Ponadto w implementacji należy wykorzystać cechy obiektowości wymienione w załączniku 2.

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:
3pkt. Implementacja świata gry i jego wizualizacji. Implementacja wszystkich 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 pkt. Implementacja możliwości dodawania organizmów do świata gry. Naciśnięcie na wolne pole powinno dać możliwość dodania każdego z istniejących w świecie organizmów.
5 pkt. Implementacja abstrakcji dla świata wraz z dwiema implementacjami. W jednej implementacji rozgrywka na kracie, w drugiej implementacji rozgrywka na planszy podzielonej na pola sześcienne (jak w grze planszowej hex).
Ponadto w implementacji należy wykorzystać cechy obiektowości wymienione w załączniku 2.

Projekt 3. Python

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

Ponadto w implementacji należy wykorzystać cechy obiektowości wymienione w załączniku 2.

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.

Id	Umiejętność	Cechy
0	Nieśmiertelność	Człowiek nie może zostać zabity. W przypadku konfrontacji z silniejszym przeciwnikiem przesuwa się na losowo wybrane pole sąsiadujące.
1	Magiczny Eliksir	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. Szczegółowy zakres wymagań technicznych w projekcie

Są to warunki konieczne do spełnienia w celu uzyskania konkretnej oceny - tj. brak któregoś z elementów wymaganych na 5pkt. spowoduje uzyskanie oceny niższej niż 5pkt. Podczas oddawania student powinien również potrafić wskazać i omówić w kodzie źródłowym miejsca w których występują wymienione dalej punkty i odpowiedzieć na związane z nimi pytania.

Klasy i obiekty

1. W projekcie należy użyć klas oraz wykorzystywać obiekty, nie jest dopuszczalne pisanie "luźnych" funkcji (poza funkcją main) (konieczne na >=3pkt)
2. Logiczny podział na przestrzenie nazw - każda przestrzeń nazw w oddzielnym module (pliku) (konieczne na 3pkt)
3. Metody które nie wykorzystują obiektu powinny być statyczne. Nie należy ich nadużywać. (konieczne na >=3pkt)
4.
5. Co najmniej jedna klasa abstrakcyjna (konieczne na >=4pkt)
6. **Hermetyzacja**

1. Wszystkie pola klas powinny być prywatne lub chronione (protected) (konieczne na >=3pkt)
2. Wybrane klasy powinny mieć metody typu get i set dla składowych lub tylko get, lub całkowity brak dostępu bezpośredniego (konieczne na >=4pkt)

Dziedziczenie

1. Przynajmniej 1 klasa bazowa po której dziedziczy bezpośrednio (w tym samym pokoleniu) kilka klas pochodnych (konieczne na >=3pkt)
2. Wielokrotne wykorzystanie kodu (kod w klasie bazowej używany przez obiekty klas pochodnych) (konieczne na >=3pkt)
3. Nadpisywanie metody klasy bazowej (konieczne na >=4pkt)
4. Jawne wywołanie metod z klasy bazowej mimo ich nadpisania w klasie pochodnej (konieczne na 5pkt)

Kompozycja

1. Implementacja Klasy (kontenera) zawierającego zestaw obiektów innej klasy. Można skorzystać z przykładowego schematu:
   (konieczne na >=3pkt)

```
class KontenerOrganizmow {
[...]
  Organizm * organizmy;
  int iloscOrganizmow;
[...]
}
```

2. Umożliwić dodawanie i usuwanie obiektów z kontenera. (konieczne na >=4pkt)
3. Zaimplementować kompozycję oraz agregację z uwzględnieniem różnicy między tymi typami relacji (konieczne na >=5pkt)

Polimorfizm

1. Implementacja i wykorzystanie kontenera do przechowywania obiektów zgodnych z abstrakcyjną klasą bazową. Przechowywanie w nim obiektów klas potomnych. Rozumienie zasady działania funkcji wirtualnych (konieczne na 4pkt).
2. Wykorzystanie metod służących do dynamicznego określania typu (konieczne na 5 pkt.)

Inne wymagania

1. Stan wszystkich obiektów (w tym kontenerów) powinien wczytywać i zapisywać się do pliku (konieczne na 4pkt)
2. Implementacja własnych konstruktorów kopiujących, destruktorów, oraz operatora przypisania (konieczne na >=5pkt)
3. Zademonstrować obsługę wyjątków (konieczne na 4pkt)
   (w szczególności użycie try, catch, throw), oraz zaimplementować przykładowe własne wyjątki (konieczne na 5pkt)

Styl programowania

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

http://geosoft.no/development/cppstyle.html (konieczne na >=3pkt)

przede wszystkim:

- 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).

Uzupełnienie

Przykład użycia szablonów i implementacji własnych wyjątków (do przeanalizowania):

```
#include "stdafx.h"
#include <iostream>
using namespace std;
//funkcja szablonowa
template<typename T>
int zapiszDoPlikuElegancko(T t, ostream& out)
{
	out << "\n*********************************\n";
	out << t;
	out << "\n*********************************\n";
	return 0;
}
class Pojazd {
	int x;
	int y;
	int vx;
	int vy;
public:
	void jedz(){
		x += vx;
		y += vy;
	}
};
class Kolo{
public:
	float srednica;
	bool przebite;
};

/* Główną zaletą poniższej klasy jest optymalizacja ilości kodu i uelastycznienie programu.
   Konkretne klasy są dynamicznie generowane na etapie kompilacji
   w każdym wariancie ilości kół jaki jest używany w programie.
   Powstaje więc faktycznie wiele klas każda idealnie dokrojona do ilości potrzebnych kół.
*/
template<const int ilosckol = 4>
class PojazdKolowy  : public Pojazd
{
```

```
	Kolo k[ilosckol];
public:
	void print(){
		cout << ilosckol;
		return;
	}
};

class Wyjatek
{
public:
	const char * tekst;
	Wyjatek(const char * tx)
	{
		tekst = tx;
	}
};
template<typename T,const int max_rozmiar = 100>
class Wypisywacz
{
	T tablica[max_rozmiar];
	int i = 0;

public:

	void dodaj(T t){
		if (i >= max_rozmiar)
			throw Wyjatek("Nie mozna juz dodawac");
		tablica[i] = t;
		i++;
	}
	void wypisz_wszystkie()
	{
		cout << "Jestem wypisywaczem wszystkiego\n";
		for (int j = 0; j < i; j++)
			cout << tablica[j] << endl;
	}
};
int _tmain(int argc, _TCHAR* argv[])
{
	try
	{
		zapiszDoPlikuElegancko(10, cout);
		zapiszDoPlikuElegancko("dziesiec\n", cout);

		PojazdKolowy<> pk;
		pk.print();
		PojazdKolowy<10> pk10;
		pk10.print();
		PojazdKolowy<10> * pointer;
		pointer = &pk10;	//OK
		//pointer = &pk;		//Błąd to są inne typy

		Wypisywacz<int> wi;
		wi.dodaj(1);
		wi.dodaj(2);
		wi.wypisz_wszystkie();

		Wypisywacz<const char*, 2> wc;

		wc.dodaj("tekst");
		wc.dodaj("tekst inny");

		wc.wypisz_wszystkie();
		wc.dodaj("oj nie uda się");		//uwaga tu bedzie wyjatek
```

```
                wc.wypisz_wszystkie();                          //tu nie dojdziemy
        }
        catch (Wyjatek& w){
                cout << "\nOj zlapany wyjatek\n" <<w.tekst;
        }
        return 0;

}
```