Celem projektu jest implementacja prostego symulatora wirtualnego świata, który ma mieć charakter dwuwymiarowej tablicy o rozmiarach NxN (domyślnie 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 1). Przy równej sile zwycięża organizm, który zaatakował. Przy uruchomieniu programu na planszy powinno się pojawić po kilka sztuk przydzielonych studentowi 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 z dziennika.

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

Należy utworzyć klasę **Świat** (Swiat) będącą kontenerem organizmów. Powinna zawierać m.in. metody:

- wykonajTure()
- rysujSwiat()

pola:

- organizmy

Należy również utworzyć abstrakcyjną klasę **Organizm**.

podstawowe pola:

- siła
- inicjatywa
- położenie (x,y) (należy zwrócić uwagę aby uniknąć możliwej redundancji - skoro obiekt organizm zawiera informację o swoim położeniu- nie powinna być ona powielona w klasie świat).
- 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.

Zaimplementuj 5 klas zwierząt (wilk, owca , jedno zwierze wymyślone przez Ciebie, 2 zwierzęta przydzielone na podstawie Twojego numeru idziennika lub inicjałów – patrz załącznik). Rodzaje zwierząt definiuje poniższa tabela.

Tabela 1. 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	żmija	2	3	brak	Ginie przy kolizji z silniejszym przeciwnikiem, ale zatruwa i zabija swojego pogromcę.
5	żółw	2	1	W 75% przypadków nie zmienia swojego położenia.	Odpiera ataki zwierząt o sile <5. **Napastnik musi wrócić na swoje poprzednie pole.**
6	ślimak	1	1	W 90% przypadków nie zmienia swojego położenia.	Niewrażliwy na ataki zwierząt o sile <2. Ma 60% szans iż pozostanie niezauważony przez zwierzęta o sile >4. **W obu przypadkach** n**apastnik przesuwa się na inne niezajęte pole.**
7	antylopa	4	4	Zasięg ruchu wynosi 2 pola.	50% szans na ucieczkę przed walką. Wówczas przesuwa się na niezajęte sąsiednie pole.
8	jeż	2	3	brak	Gdy ginie, tak mocno kaleczy swojego pogromcę, że ten nie może się ruszać przez kolejne dwie tury.
9	lew	11	7	brak	Król zwierząt: żadne zwierzę o sile < 5 nie ośmieli się wejść na pole zajmowane przez lwa
10	mysz	1	6	brak	Potrafi uciec napastnikowi na sąsiednie pole jeśli jest wolne. Nie działa gdy wrogiem jest żmija.
11	komar	1	1	+1 do inicjatywy i +1 do siły za każdego sąsiadującego komara	Jeśli zostanie pokonany, ma 50% szans na przeżycie (wraca na poprzednie pole).
12	Leniwiec	2	1	Nigdy nie przemieszcza się dwa razy pod rząd w kolejnych turach	brak

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 3 klasy roślin (trawa oraz 2 rośliny przydzielone na podstawie Twojego numeru z dziennika lub inicjałów – patrz załącznik). Rodzaje roślin definiuje poniższa tabela.

Tabela 2. 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
koka	0	brak	Zwierze, które zjadło tę roślinę w następnej kolejce ma dodatkowy ruch.
guarana	0	brak	Zwiększa siłę zwierzęcia, które zjadło tę roślinę, o 3.
wilcze jagody	0	brak	Zwierze, które zjadło tę roślinę, ginie.
cierń	2	Próby rozprzestrzeniania się zawsze kończą się sukcesem.	brak

Stwórz klasę **Świat** zawierającą dwuwymiarową tablicę wskaźników lub referencji (w zależności od stosowanego języka programowania) na 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.

Pythom

Wizualizację nalezy wykonać za pomocą PyGame oraz PyGame GUI. Organizmy mają być przedstawione za pomocą obrazków. Aplikacja ma posiadać kilka funkcjonalności na wyższe oceny. Głownym przyciskiem jest Następna Tura oraz Następny Organizm. Powodują one wykonanie akcji przez wszystkie organizmy lub pojedynczo. Kawałek okna ma przedstawiać Tekst, który jest wynikiem zapisów Narratora. Dodaje on tam wszystkie wydarzenia, które wydarzyły się podczas trwania tury np.: "Wilk na polu 3,4 zjadł Owca na polu 3,5 i przeszedł na jego pole". Ma się on czyścić po zakończeniu tury przez wszystkie organizmy. Przycisk zapis/wczytaj - pozwala na uruchamianie wcześniej zapisanych stanów symulacji. Przycisk Dodaj... ma pozwalać na dodanie dowolnego organizmu na planszę podczas symulacji.

Punktacja:

3 -> Implementacja świata gry i jego wizualizacji. Implementacja wszystkich przydzielonych gatunków zwierząt, bez rozmnażania. Implementacja wszystkich przydzielonych gatunków roślin, bez rozprzestrzeniania.

4 -> rozmnażanie się zwierząt i rozprzestrzenianie się roślin.

5 -> Implementacja możliwości zapisania do pliku i wczytania z pliku stanu wirtualnego świata.

6 -> Implementacja możliwości dodawania organizmów do świata gry. Naciśnięcie na wolne pole powinno dać możliwość dodania każdego z istniejących w świecie organizmów.

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

Załącznik 1. Sposób przydziału organizmów poszczególnym dzieciaczkom.

Przydział zwierząt i roślin jest zdeterminowany numerem w dzienniku oraz inicjałami autora.

Przydział zwierząt jest realizowany w następujący sposób:

$$ID_1 = (X \bmod 5) + 3,$$

$$ID_2 = (Y \bmod 5) + 8$$

gdzie:

ID_1 – id (wg tabeli 1) pierwszego ze zwierząt

ID_2 – id (wg tabeli 1) drugiego ze zwierząt

X – przedostatnia cyfra numeru w dzienniku (jeśli jednocyfrowa => wpisz 0)

Y– ostatnia cyfra numeru z dziennika

```
if  ($ID_1 == ID_2$)  then
    $ID_2 = [(Y+1) \bmod 5] + 8$;
end;
```

Przydział roślin jest uzależniony do inicjałów pochodzących od imienia i nazwiska autora wg następującej tabeli:

Tabela 3. Zasada przydziału roślin na podstawie inicjałów imienia i nazwiska autora.

Litery od A do D	mlecz
Litery od E do H	koka
Litery od I do M	guarana
Litery od N do P	wilcze jagody
Litery od R do Z	cierń

UWAGA!

Osoby posiadające inicjały składające się z takich samych liter powinny zaimplementować roślinę która wynika z ich inicjałów oraz następną w kolejności (lub poprzednią, jeżeli według algorytmu przypada im ostatnia roślina w tabeli).

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 6pkt. spowoduje uzyskanie oceny niższej niż 6pkt. 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) -1
2. Logiczny podział na przestrzenie nazw - każda przestrzeń nazw w oddzielnym module (pliku) (konieczne na 3pkt) -1

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 wraz z wywołaniem jej w implementacji klasy pochodnej (konieczne na >=4pkt)

Kompozycja

1. Klasa (kontener) zawierający zestaw obiektów innej klasy wg przykładowego schematu: (konieczne na >=3pkt)
```
class WirtualnySwiat {
[...]
  Organizm * organizmy;
  int iloscOrganizmow;
[...]
}
```
2. Dla powyższego przykładu umożliwić co najmniej dodawanie i usuwanie obiektów z tablicy. (konieczne na >=5pkt)
3. Dla powyższego przykładu umożliwić transfer obiektu pomiędzy różnymi kontenerami bez tworzenia nowego obiektu (konieczne na >=5pkt)

Polimorfizm

1. Implementacja tablicy obiektów klasy macierzystej, w której będą przechowywane obiekty klas potomnych. Wywołanie tej samej metody na każdym polu tej tabeli (konieczne na 3pkt)

Inne wymagania

1. Stan wszystkich obiektów (w tym kontenerów) powinien wczytywać i zapisywać się do pliku (konieczne na 5pkt)
2. Implementacja własnych konstruktorów kopiujących implementujących kopiowanie jeśli zwykły konstruktor kopiujący -domyślny - nie wystarcza (konieczne na >=4pkt)
3. Zaimplementować i zademonstrować własne wyjątki (konieczne na 5pkt)
   (w szczególności użycie try, catch, throw)