**Ministerul Educaţiei și Cercetării al Republicii Moldova**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică și Microelectronică**

**RAPORT**

Lucrarea de laborator nr.4

*la Programarea Orientată pe Obiecte*

A efectuat:

st. gr. TI-231 Calaev Vitalie

A verificat:

asist. univ. Coșeru Cătălin

Chişinău - 2024

**Lucrare de laborator nr.4**

**Tema:** Simulare Ecosistem.

**Scopul lucrării:**

1. Simularea a unui ecosistem complex în care animalele și plantele interacționează între ele.
2. Implementarea codului.
3. Testarea codului.
4. Documentația proiectului

**Documentația proiectului - Ecosistem Simulat**

**Descrierea Claselor și Ierarhiilor**

*EntitateEcosistem* (Clasă Abstractă), această clasă este baza pentru toate entitățile din ecosistem. Conține caracteristici generale precum nume, energie, poziție și rata de supraviețuire.

1. Atribute:
   1. nume: Numele entității.
   2. energie: Cantitatea de energie a entității.
   3. pozitie: Coordonatele entității pe hartă (tuple).
   4. rata\_supravietuire: Probabilitatea de supraviețuire.
2. Metode:
   1. actioneaza(self, dimensiune\_harta=None): Definește comportamentul entității la fiecare pas de simulare.

*Planta* (Derivă din EntitateEcosistem), reprezintă plantele din ecosistem. Energia lor crește automat la fiecare pas de simulare.

1. Atribute:
   1. rata\_crestere: Cantitatea de energie câștigată la fiecare pas.
2. Metode:
   1. actioneaza(self, dimensiune\_harta=None): Crește energia plantei.

*Animal* (Clasă Abstractă)ste baza pentru toate animalele din ecosistem.

1. Atribute:
   1. viteza: Numărul maxim de pași pe care animalul îi poate face la fiecare mișcare.
   2. tip\_hrana: Tipul de hrană consumată (plante, animale, ambele).
2. Metode:
   1. deplaseaza(self, dimensiune\_harta): Animalul se mișcă aleator în limitele hărții.
   2. mananca(self, prada): Definește comportamentul de consum.
   3. actioneaza(self, dimensiune\_harta): Definește acțiunile specifice.

*Erbivor* (Derivă din Animal), reprezintă animalele care se hrănesc doar cu plante.

1. Metode:
   1. mananca(self, prada): Consumul de plante.
   2. actioneaza(self, dimensiune\_harta): Deplasare și căutare de hrană.

*Carnivor* (Derivă din Animal), reprezintă animalele care se hrănesc doar cu alte animale.

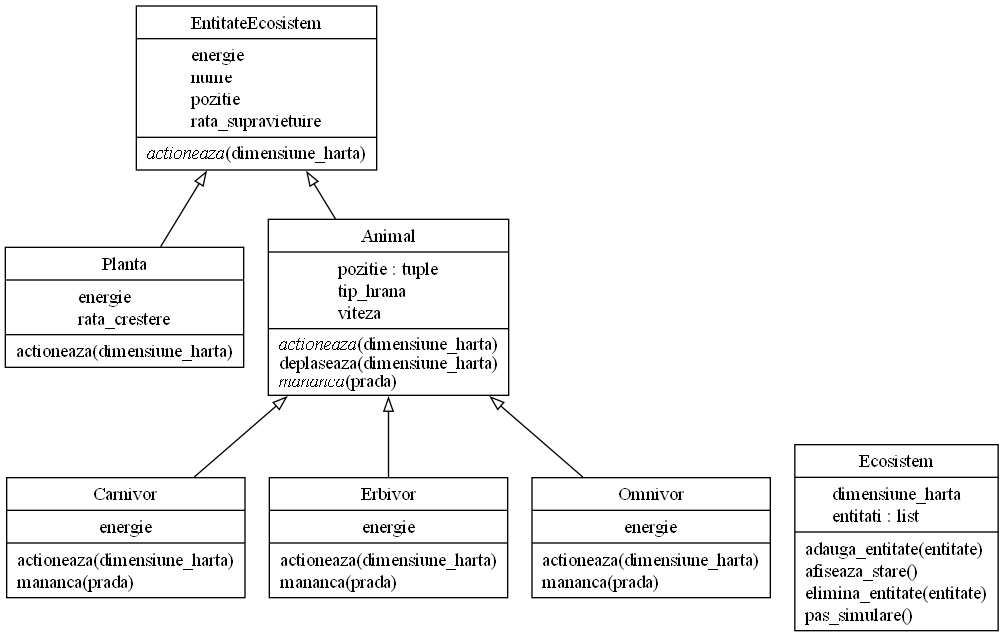
1. Metode:
   1. mananca(self, prada): Consumul altor animale.
   2. actioneaza(self, dimensiune\_harta): Deplasare și vânătoare.

*Omnivor* (Derivă din Animal), reprezintă animalele care consumă atât plante, cât și alte animale.

1. Metode:
   1. mananca(self, prada): Consumul mixt (plante sau animale).
   2. actioneaza(self, dimensiune\_harta): Deplasare și consum mixt.

*Ecosistem*. Gestionarea întregului ecosistem, inclusiv a entităților și a interacțiunilor dintre ele.

1. Atribute:
   1. dimensiune\_harta: Dimensiunea hărții (o hartă pătrată).
   2. entitati: Lista entităților din ecosistem.
2. Metode:
   1. adauga\_entitate(self, entitate): Adaugă o entitate în ecosistem.
   2. elimina\_entitate(self, entitate): Elimină o entitate din ecosistem.
   3. pas\_simulare(self): Execută un pas de simulare.
   4. afiseaza\_stare(self): Afișează starea curentă a ecosistemului.

**Diagrama UML al proiectului**

### **Scenarii de Utilizare**

1. Adăugare de entități în ecosistem:

Python code:

eco = Ecosistem(dimensiune\_harta=5)

planta = Planta("Planta1", energie=10, pozitie=(0, 0), rata\_crestere=5)

eco.adauga\_entitate(planta)

1. Simulare simplă a ecosistemului:Python code:  
    for \_ in range(5): # 5 pași de simulare

eco.pas\_simulare()

eco.afiseaza\_stare()

1. Interacțiuni între entități (erbivor care mănâncă o plantă):Python code:  
    planta = Planta("Planta2", energie=10, pozitie=(1, 1), rata\_crestere=5)

iepure = Erbivor("Iepure", energie=15, pozitie=(1, 1), viteza=1, tip\_hrana="plante")

eco.adauga\_entitate(planta)

eco.adauga\_entitate(iepure)

eco.pas\_simulare() # Iepurele mănâncă planta.

### **Dificultățile Întâlnite și Soluțiile Adoptate**

1. Problema: Animalele ieșeau din limitele hărții.
   1. Soluție: Am modificat metoda deplaseaza astfel încât să limiteze coordonatele între 0 și dimensiune\_harta - 1.
2. Problema: Metoda actioneaza era neclară și incompletă.
   1. Soluție: Am implementat logică specifică pentru fiecare tip de entitate (creștere pentru plante, deplasare și hrănire pentru animale).
3. Problema: Gestionarea simultană a interacțiunilor între entități.
   1. Soluție: Am verificat pozițiile entităților și am implementat logica de consum în metoda pas\_simulare.
4. Problema: Redundanță în tipurile de hrană.
   1. Soluție: Am creat o metodă abstractă mananca în clasa Animal și am personalizat-o pentru erbivore, carnivore și omnivore.

**Concluzie**

În cadrul acestui laborator, am implementat un ecosistem simulând interacțiuni între plante și animale folosind principii de programare orientată pe obiecte (OOP). Am creat o ierarhie de clase pentru entități, cu comportamente specifice pentru erbivore, carnivore și omnivore. Am gestionat deplasarea entităților pe hartă și interacțiunile între prădători și prăzi. Dificultățile întâmpinate au fost legate de gestionarea corectă a deplasărilor și a coliziunilor, dar au fost rezolvate prin ajustări funcționale. Laboratorul a consolidat înțelegerea principiilor OOP și aplicarea acestora într-un proiect practic.

**Codul**

from abc import ABC, abstractmethod

import random

class EntitateEcosistem(ABC):

def \_\_init\_\_(self, nume, energie, pozitie, rata\_supravietuire):

self.nume = nume

self.energie = energie

self.pozitie = pozitie # (x, y)

self.rata\_supravietuire = rata\_supravietuire

@abstractmethod

def actioneaza(self, dimensiune\_harta=None):

"""Metodă abstractă pentru acțiunea entității."""

pass

class Planta(EntitateEcosistem):

def \_\_init\_\_(self, nume, energie, pozitie, rata\_crestere):

super().\_\_init\_\_(nume, energie, pozitie, rata\_supravietuire=1.0)

self.rata\_crestere = rata\_crestere # energie castigata la fiecare pas

def actioneaza(self, dimensiune\_harta=None):

"""Crește energia plantei."""

self.energie += self.rata\_crestere

print(f"{self.nume} a crescut și acum are energie {self.energie}.")

class Animal(EntitateEcosistem):

def \_\_init\_\_(self, nume, energie, pozitie, viteza, tip\_hrana):

super().\_\_init\_\_(nume, energie, pozitie, rata\_supravietuire=0.5)

self.viteza = viteza # numarul maxim de pasi pe care-l poate face

self.tip\_hrana = tip\_hrana # "plante", "animale", "ambele"

def deplaseaza(self, dimensiune\_harta):

"""Se deplasează într-o direcție aleatorie în limita hărții."""

dx = random.randint(-self.viteza, self.viteza)

dy = random.randint(-self.viteza, self.viteza)

x\_nou = max(0, min(self.pozitie[0] + dx, dimensiune\_harta - 1))

y\_nou = max(0, min(self.pozitie[1] + dy, dimensiune\_harta - 1))

self.pozitie = (x\_nou, y\_nou)

print(f"{self.nume} s-a deplasat la poziția {self.pozitie}.")

@abstractmethod

def mananca(self, prada):

"""Metodă abstractă pentru mâncat."""

pass

@abstractmethod

def actioneaza(self, dimensiune\_harta):

pass

class Erbivor(Animal):

def mananca(self, prada):

if isinstance(prada, Planta):

self.energie += prada.energie

prada.energie = 0

print(f"{self.nume} a mâncat {prada.nume} și a câștigat energie.")

else:

print(f"{self.nume} nu poate mânca {prada.nume}.")

def actioneaza(self, dimensiune\_harta):

"""Acțiunea erbivorului: deplasare și consum de hrană."""

self.deplaseaza(dimensiune\_harta)

print(f"{self.nume} analizează mediul și caută hrană.")

class Carnivor(Animal):

def mananca(self, prada):

if isinstance(prada, Animal) and prada.energie > 0:

self.energie += prada.energie

prada.energie = 0

print(f"{self.nume} a vânat și a mâncat {prada.nume}.")

else:

print(f"{self.nume} nu poate mânca {prada.nume}.")

def actioneaza(self, dimensiune\_harta):

"""Acțiunea carnivorului: deplasare și vânătoare."""

self.deplaseaza(dimensiune\_harta)

print(f"{self.nume} patrulează zona în căutarea prăzii.")

class Omnivor(Animal):

def mananca(self, prada):

if isinstance(prada, (Planta, Animal)) and prada.energie > 0:

self.energie += prada.energie

prada.energie = 0

print(f"{self.nume} a mâncat {prada.nume}.")

else:

print(f"{self.nume} nu poate mânca {prada.nume}.")

def actioneaza(self, dimensiune\_harta):

"""Acțiunea omnivorului: deplasare și consum mixt."""

self.deplaseaza(dimensiune\_harta)

print(f"{self.nume} explorează mediul și caută hrană mixtă.")

class Ecosistem:

def \_\_init\_\_(self, dimensiune\_harta):

self.dimensiune\_harta = dimensiune\_harta

self.entitati = []

def adauga\_entitate(self, entitate):

self.entitati.append(entitate)

def elimina\_entitate(self, entitate):

self.entitati.remove(entitate)

def pas\_simulare(self):

for entitate in self.entitati:

if isinstance(entitate, Animal):

entitate.actioneaza(self.dimensiune\_harta)

else:

entitate.actioneaza()

# Verifica interactiunile

for prada in self.entitati[:]:

for vanator in self.entitati:

if isinstance(vanator, Animal) and vanator != prada:

if vanator.pozitie == prada.pozitie:

vanator.mananca(prada)

if prada.energie <= 0:

self.elimina\_entitate(prada)

print(f"{prada.nume} a murit.")

def afiseaza\_stare(self):

print("Starea ecosistemului:")

for entitate in self.entitati:

print(f"- {entitate.nume}: energie {entitate.energie}, poziție {entitate.pozitie}")

eco = Ecosistem(dimensiune\_harta=10)

planta1 = Planta("Planta1", energie=10, pozitie=(0, 0), rata\_crestere=5)

planta2 = Planta("Planta2", energie=8, pozitie=(1, 1), rata\_crestere=4)

eco.adauga\_entitate(planta1)

eco.adauga\_entitate(planta2)

iepure = Erbivor("Iepure", energie=15, pozitie=(0, 1), viteza=2, tip\_hrana="plante")

lup = Carnivor("Lup", energie=20, pozitie=(1, 0), viteza=3, tip\_hrana="animale")

gorila = Omnivor("Gorila", energie=26, pozitie=(2, 3), viteza=1, tip\_hrana="ambele")

eco.adauga\_entitate(iepure)

eco.adauga\_entitate(lup)

eco.adauga\_entitate(gorila)

# Simulare

for \_ in range(5): # Simuleaza 5 pasi

print("\n--- Pas nou ---")

eco.pas\_simulare()

eco.afiseaza\_stare()