Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №2

Розробка інтелектуальної інформаційної систем на основі технології моделювання розвитку природних систем

Виконав
студент групи ІТ-41ф

Новиков Д. М.

Перевірив:

доц. каф. ІСТ
Кравець П.І.

Київ 2025

Мета роботи: навчитися створювати інтелектуальні інформаційні системи на основі технології розвитку природних систем для моделювання процесів в економіці та живій природі, а також для моделювання взаємодії різноманітних об'єктів в конфліктних ситуаціях.

Завдання роботи:

1. Ознайомитися з літературою та основними теоретичними відомостями за темою роботи.
2. Розробити програмне забезпечення, що моделює штучне життя (модель харчового ланцюгу). При розробці програмного забезпечення врахувати твердження, наведені в п.2.3.2.1-2.3.2.5.
3. Штучний світ моделюється за допомогою сітки розмірності NxN.
4. Існує всього 3 типи агентів:
    a. рослини (при цьому в процесі моделювання, коли одну рослину з'їдають, має моделюватися поява нової рослини);
    b. травоїдні: вони мають становити максимально 50% від максимального можливого значення агентів і не можуть бути в кількості, меншій за 25% від максимально можливого значення агентів. При цьому травоїдні можуть їсти рослини, в той же час вони є їжею для 3-го типу агентів - м'ясоїдних;
    c. м'ясоїдні: обмеження на їхню кількість такі ж, як і для травоїдних.
5. Агенти, що моделюють травоїдних та м'ясоїдних, мають бути реалізовані у вигляді структури, що містить наступні поля:
    a. тип;
    b. енергія;
    c. вік - кількість ітерацій, на протязі яких жив агент;
    d. покоління: спочатку встановлюється в 1, збільшується, якщо агент є нащадком іншого (тобто збільшується в процесі розмноження);
    e. координата X;
    f. координата Y;
    g. напрям погляду агенту - може бути північним, східним, західним або південним;
    h. входи нейромережі: на входи поступають кількості інших агентів відповідно до їх положення відносно даного агенту. Тобто на вхід 1 поступає кількість травоїдних на передньому плані, на вхід 2 - кількість м'ясоїдних на передньому плані, на вхід 3 - кількість рослин на передньому плані, на входи 4-6 - кількості ті ж агентів, що і для входів 1-3, але вже ліворуч, 7-9 - праворуч, 10-12 - у безпосередній близькості;
    i. ваги нейромережі, які вибираються випадковим чином один раз при ініціалізації моделювання;
    j. зміщення для нейромережі - вибираються випадковим чином при ініціалізації;
    k. 4 виходи нейромережі, значення яких розраховуються за наведеною в основних теоретичних відомостях формулою. Значення виходів відповідають можливим діям агенту: повернути ліворуч (тобто лише

змінити напрям погляду), повернути праворуч, рух вперед відповідно до напряму, з'їсти іншого агенту. При цьому, якщо було прийнято рішення з'їсти іншого агента, то необов'язково хтось буде з'їденим, оскільки дане рішення може бути помилковим. Обирається та дія, значення якої є найбільшим.

6. При моделюванні штучного життя вважати, що при поїданні рослини отримується енергія, вдвічі менша за поїдання травоїдного;
7. Зберігати наступну статистику:
    a. кількість агентів відповідних типів на даній ітерації;
    b. максимальний вік за весь час моделювання агентів відповідних типів;
    c. кількість розмножень агентів для кожного типу.
8. Обґрунтовано сформувати набір даних для тестування розробленого програмного забезпечення. Виконати тестування. Результати тестування оформити в вигляді таблиць та графіків.
9. Оформити звіт з роботи.

Хід роботи:

Інтерфейс програми представлено (Рис. 1):

- Блок параметрів:
    - Map size (N) – розмір мапи по одній осі (N*N);
    - Plants count (forest only) – початкова кількість рослин у лісі (центральна зона, що займає ≈75% площі мапи; по краях рослини не з'являються);
    - New plants per eat (0=off) – скільки нових рослин спавниться у лісі кожного разу, коли травоїдний з'їдає рослину;
    - Herbivore count – кількість травоїдних агентів на старті;
    - Predator count – кількість агентів-хижаків на старті:
    - Iterations (days) – кількість ітерацій;
    - Disable agent limits (25-50% of $N^2/5$) – прапорець що вимикає ліміти на кількість агентів. Якщо вимкнено, то початкова кількість травоїдних/хижаків автоматично обмежується діапазоном 25-50% від ($N^2/5$); фактичні значення буде показано в полях після запуску;
    - Кнопки керування:
        - Run algorithm – запустити симуляцію з поточними параметрами.
        - Previous – перейти до попередньої збереженої ітерації (перегляд результатів).
        - Next – перейти до наступної збереженої ітерації (перегляд результатів).
- Блок результатів (динамічний):
    - Поточну ітерацію, кількість травоядних та хижаків на даній операції;
    - Візуалізацію рішення у поточній ітерації:
        - тло (порожньо) – жовтий;
        - рослини – зелений;
        - травоїдні – синій;

- хижаки – червоний.

- Графік “Populations per step” – динаміка кількості рослин, травоїдних і хижаків по кроках симуляції. Вертикальна штрихова лінія показує поточний переглянутий крок (синхронізована з мапою та заголовком):
- Слайдер ітерації – швидка навігація за всіма кроками симуляції; рух слайдера оновлює мапу, заголовок і вертикальну лінію графіка;
- Підсумкова статистика (у блоці параметрів під кнопками):
  - Max age H/P – найбільший вік, якого досягав агент відповідного типу за всю симуляцію;
  - Reprod H/P – загальна кількість подій розмноження кожного типу.

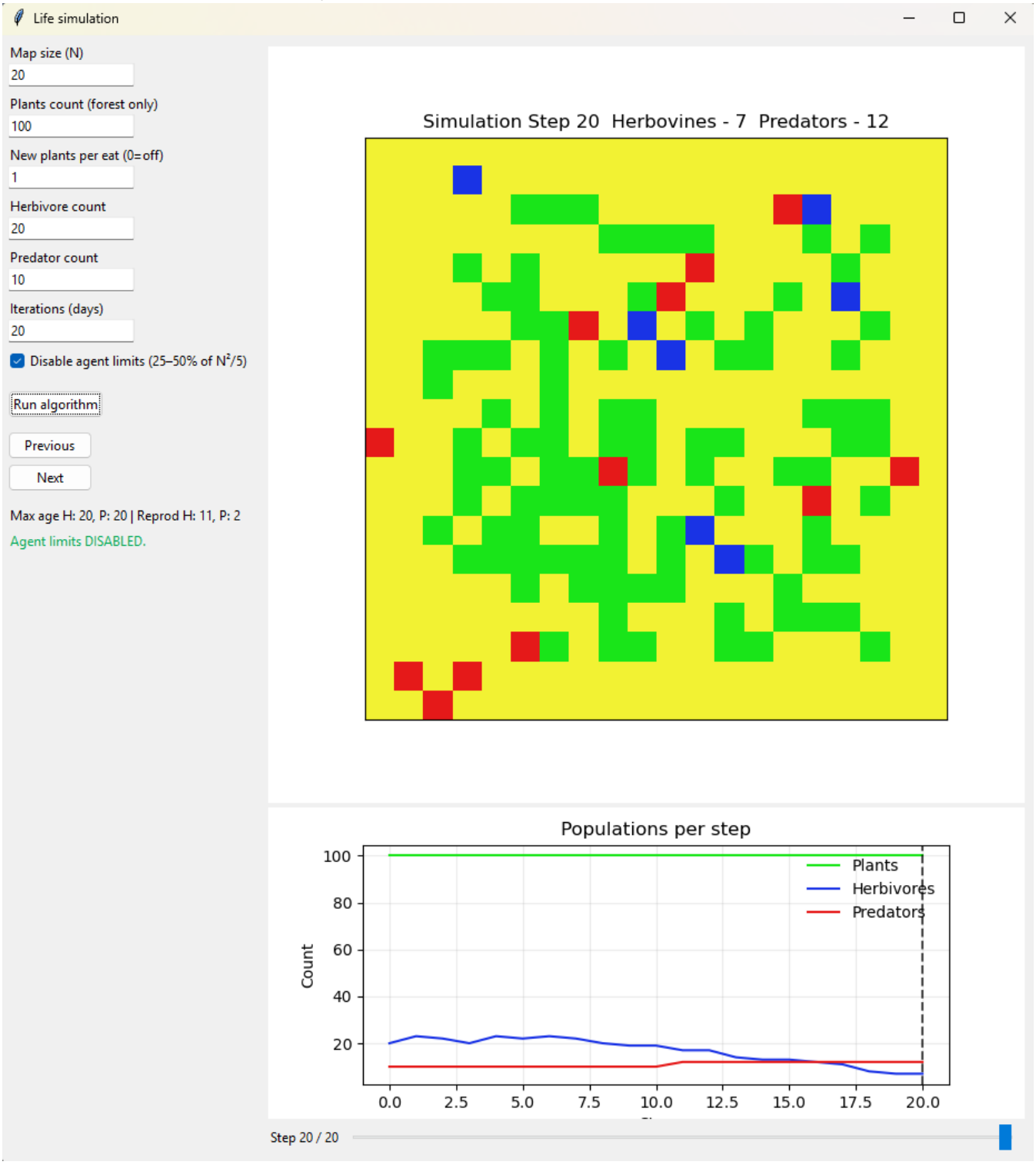


Рис 1 – Інтерфейс програми

Розроблена система працює наступним чином:

1. Представлення світу
   Мапа: квадратна ґратка N*N. Краї – “пустеля”, у центрі – ліс: квадрат, що займає ≈75% довжини по кожній осі (по краях рослин немає);
   Агент (травоїдний або хижак).
2. Ініціалізація
   - Обчислюємо координати лісу (центральні 75%);
   - Розкладаємо Plants count довільно всередині лісу по вільних клітинках;
   - Розміщуємо Herbivore count і Predator count у випадкових порожніх клітинках шару тварин (рослини не заважають);
   - Початкові стани:
     - Енергія: травоїдні 10, хижаки 20;
     - Вік: 0, Покоління: 1.
3. Поле зору та тактика:
   Кожна тварина “бачить” до 2 клітин у всіх напрямках: підраховуються кількості рослин/агентів у секторах front/left/right/near;
   Травоїдний: якщо в радіусі <= 2 є рослина – обирає напрям до найближчої рослини; інакше – випадкове повертання/крок;
   Хижак: якщо в радіусі <= 2 є травоїдний – іде до найближчого; якщо травоїдний прямо попереду – виконує стрибок/атаку на 1 клітинку вперед;
   Хижак може проходити через клітинки з рослинами (рослини його не блокують).
4. Оновлення стану за крок (доба):
   Порядок дій для всієї популяції (одна ітерація):
   - ОРІЄНТАЦІЯ (безкоштовна): вибір бажаного напряму за правилами з п.3;
   - РУХ: рівно на 1 клітинку;
     - Якщо попереду агент того ж типу – крок блокується;
     - Для хижака: якщо попереду травоїдний – атака: хижак входить у клітинку жертви в цей самий крок.
   - ЇЖА:
     - Травоїдний: якщо після руху попереду є рослина – з’їдає її, +10 енергії; рослину прибираємо;
     - Для кожної з’їденої рослини додаємо New plants per eat (NP) нових рослин у випадкові порожні клітинки всередині лісу (якщо NP=0 – респаун вимкнено);
     - Хижак: під час атаки миттєво з’їдає травоїдного, +20 енергії (жертва зникає).
   - РОЗМНОЖЕННЯ (природний відбір): якщо енергія агента >= 90% умовного максимуму (для прикладу можна брати MaxH=80, MaxP=40):
     - Знаходимо вільного сусіда; створюємо “дитину” з generation+1;

        - Енергія батька ділиться навпіл: половина лишається батьку, половина – дитині;
        - Мозок дитини – від батька з малою мутацією ваг;
        - Лічильник Reprod для типу збільшується.
    - МЕТАБОЛІЗМ/ВІК/СМЕРТЬ:
        - Щодоби кожна тварина втрачає 1 енергію, травоїдна – 2 енергії;
        - age += 1; якщо energy <= 0 – агент помирає (клітинка звільняється).
    - СТАТИСТИКА:
        - Записуємо на кожному кроці: (plants, H, P), максимальний вік за типами, кількість розмножень;
5. Респаун рослин (тільки за поїдання):
    - Єдиний механізм появи нових рослин – параметр New plants per eat: за кожну з'їдену рослину додається NP нових у межах лісу;
    - Якщо вільних клітин у лісі немає – респаун пропускається.

Мета експериментів: перевірити логіку моделі, стабільність UI та екологічну динаміку. Кожен сценарій містить мету, налаштування, послідовність кроків, метрики/перевірки й критерії приймання.

Показники, що фіксуються:

1. Кількість травоїдних;
2. Кількість хижаків;
3. Max age H/P;
4. Avg – середнє значення для усіх повторів.

Сценарії експериментів:

1. Коректність моделі (енергія, атака, розмноження, розташування лісу):
    - Мета: верифікувати модель – метаболізм, рух, атаку, розмноження, геометрія лісу.
    - Налаштування (серія коротких контрольованих запусків, 20 кроків, N=10..16):
        - 1) Метаболізм/голод: Plants=0, Respawn=0, H=3, P=20, iterations=50 => хижаки вбивають травоїдних, більшість вмирає через брак енергії;
        - 2) Розмноження: Plants=150 (у лісі), Respawn=3, H=1, P=0 => енергія H досягає порогу, з'являється нащадок;
    - Кроки:
        - Відстежити щоденне зменшення енергії (H-2, P-1), досягнення <=0 => видалення агента;
        - Після накопичення енергії H => 90% MAX => дитина поруч; енергія батька ділиться; generation+1; Reprod H зростає.
        - Перевірити, що рослини ніколи не з'являються на краях; їхня кількість <= площі центрального "лісу".

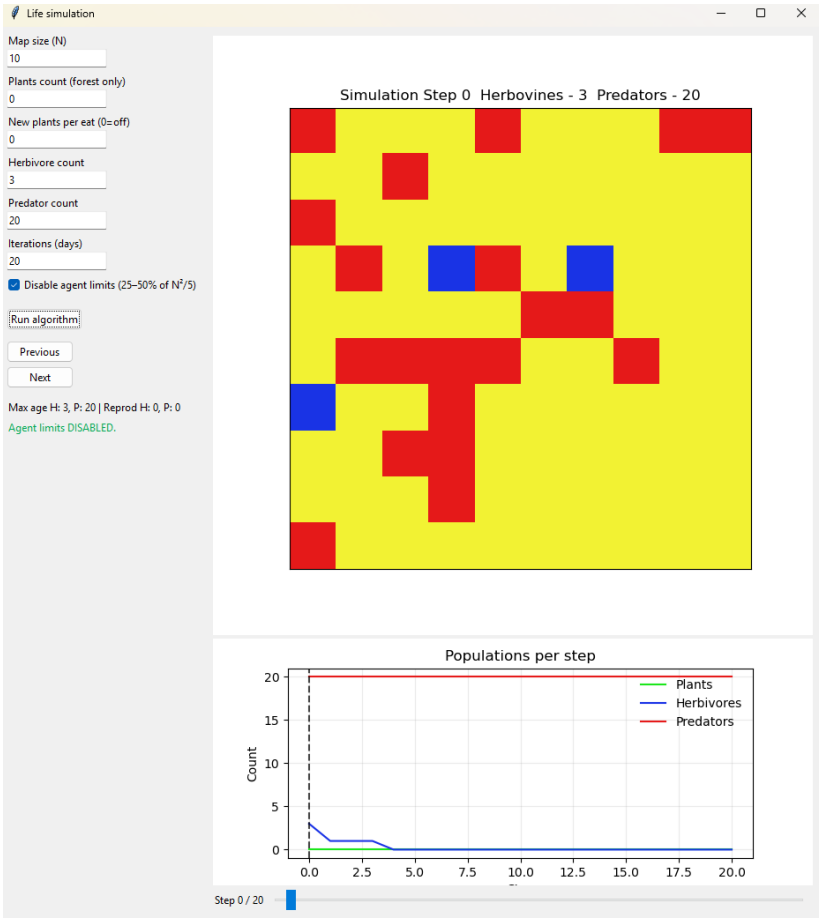
- Метрики/перевірки:
  - Втрата енергії за добу відповідає LOSS_*; енергія не стає від’ємною; смерть очищає клітинку;
  - Атака відбувається в той самий крок; немає “подвійного руху” P цього ж дня;
  - При створенні дитини generation зростає, Reprod-лічильник збільшується;
  - У жодному кроці рослина не виходить за межі 75% центра.
- Критерії оцінки:
  - Візуалізація та заголовок узгоджені; графік підкріплює чисельність.

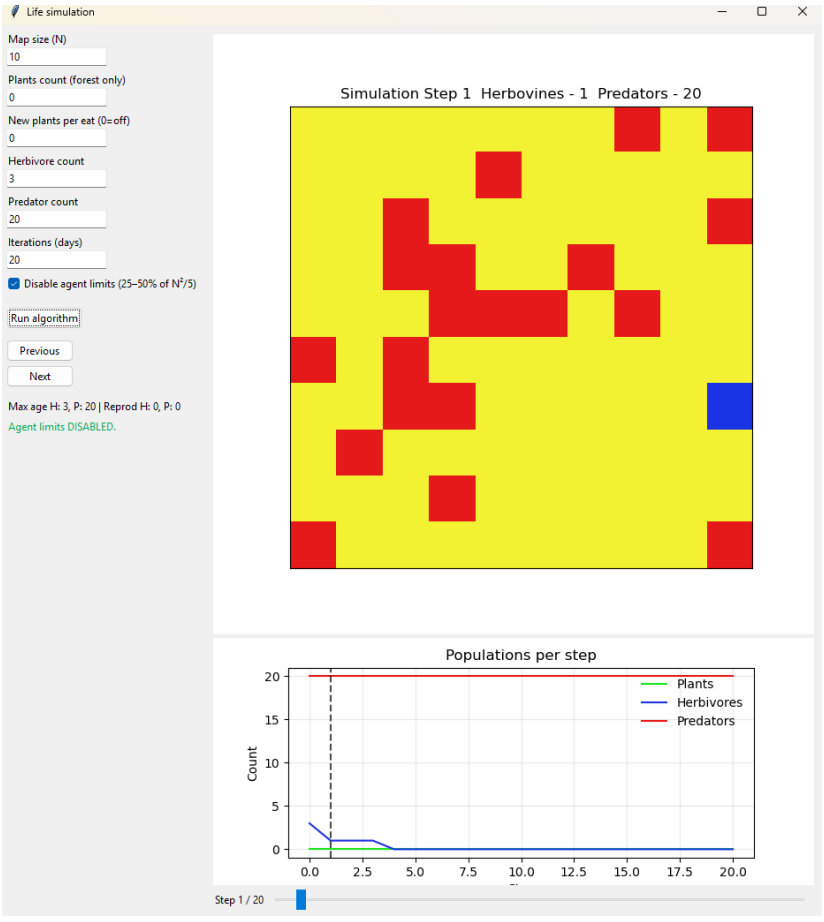
2. Дефіцит ресурсу:
   - Мета: модель дає правдоподібні траєкторії популяцій при дефіциту ресурсів;
   - Налаштування: 60 кроків, N=64, Plants=1500, Respawn=0, H=120, P=120;
   - Очікування: H та P падають; H зникають швидше/частіше за P.
3. Стабільний ресурс:
   - Мета: модель дає правдоподібні траєкторії популяцій при стабільній кількості ресурсів;
   - Налаштування: 100 кроків, N=64, Plants=2200, Respawn=1, H=180, P=45;
   - Очікування: H спочатку зростає, потім повністю зникає; P збільшуються з плином часу, а потім вимирає через брак здобичі.
4. Перенасичення хижаками:
   - Мета: модель дає правдоподібні траєкторії популяцій при великій кількості хижаків;
   - Налаштування: 60 кроків, N=64, Plants=2200, Respawn=0, H=200, P=400;
   - Очікування: H різко спадає в перші 10–20 кроків; потім P падають через брак здобичі (крива “хвилею”).

План запусків:
Для кожного сценарію:

- виконати 5 запусків;
- зберегти: кількість H/P, max age H/P, reprod H/P, Avg.

Приклад роботи:
1.1) Коректність моделі – метаболізм/голод:
N=10; P=0; NP=0; H=3; P=20; I=20



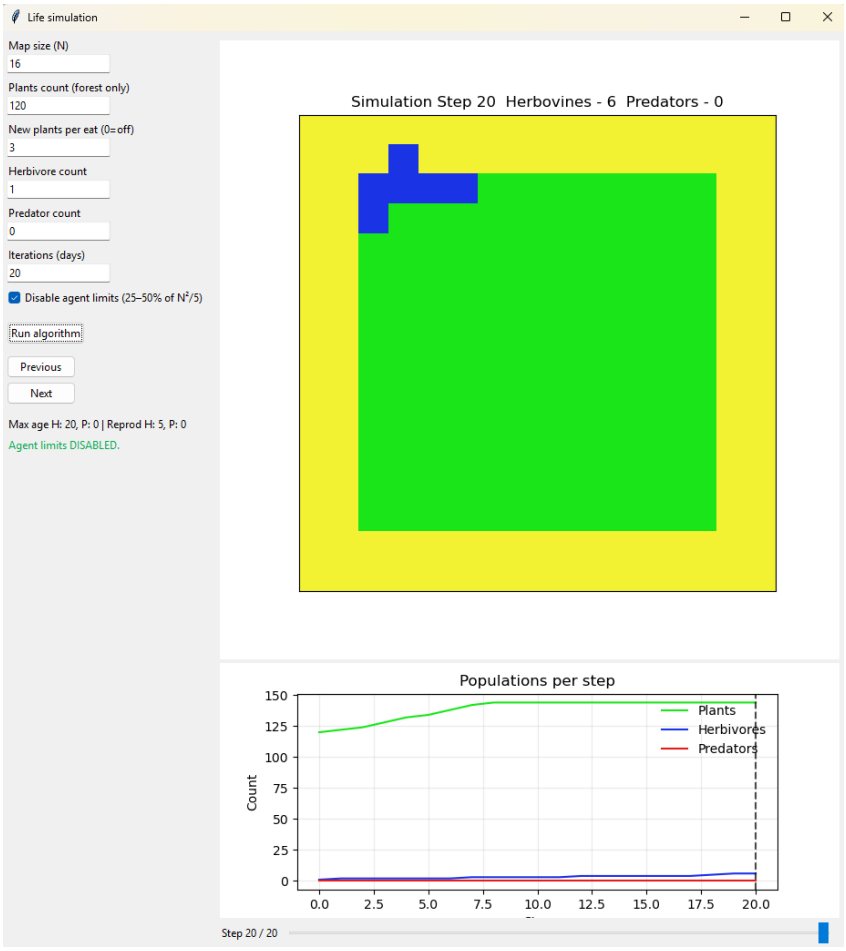


Iteration	H	P	Max age H	Max age P
1	0	20	3	20
2	0	20	2	20
3	0	20	2	20
4	0	20	1	20

5	0	20	3	20
Avg	0.00	20.00	2.20	20.00

У тесті без рослин (переспавн – 0) травоїдні швидко вимирають, а хижаки з нижчим метаболізмом і достатнім стартовим запасом енергії доживають до кінця 20 днів. Поведінка повністю відповідає правилам моделі – механіка метаболізму/голоду працює коректно для цієї конфігурації.

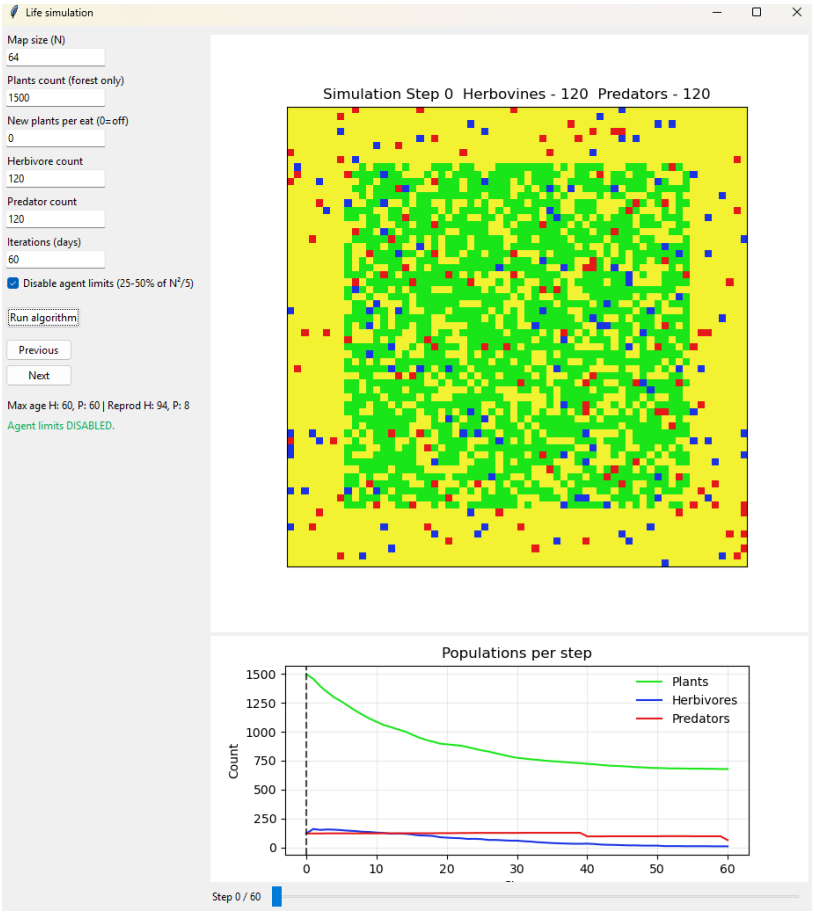
1.2) Коректність моделі – розмноження:
N=16; P=120; NP=3; H=1; P=0; I=20

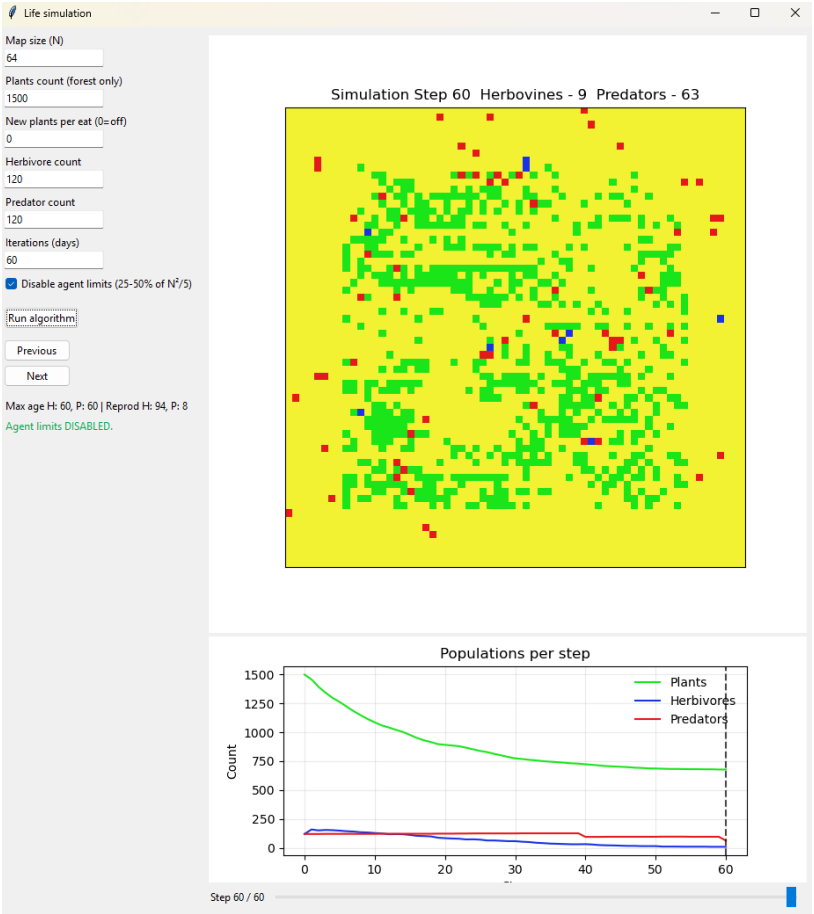


Iteration	H	P	Max age H	Max age P
1	6	0	20	0
2	9	0	20	0
3	8	0	20	0
4	8	0	20	0
5	5	0	20	0
Avg	7.20	0.00	20.00	0.00

За умов багатого корму (120 рослин, респавн NP=3) і відсутності хижаків один травоїдний стабільно розмножується (середнє 7.2 особини на 20-му кроці), а Max age свідчить, що лінія безперервно жила весь період. Модель відтворення працює коректно: досягнення порога енергії => поява нащадку поруч => поділ енергії, що забезпечує помірне зростання популяції без вибухового перегріву.

2) Дефіцит ресурсу:
N=64; P=1500; NP=0; H=120; P=120; I=60



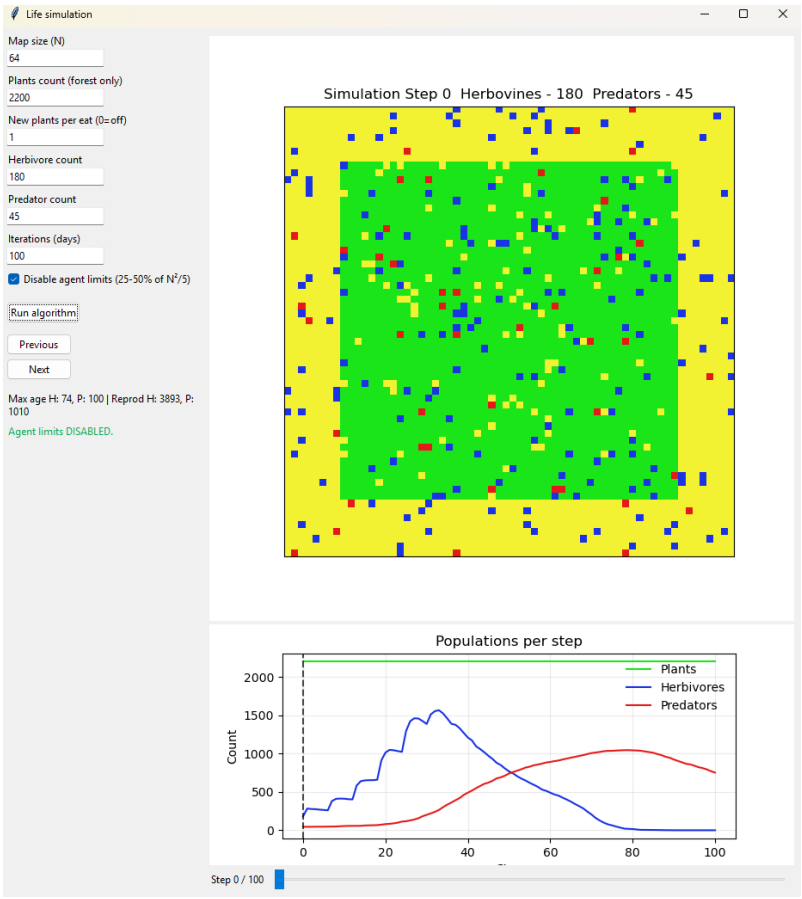


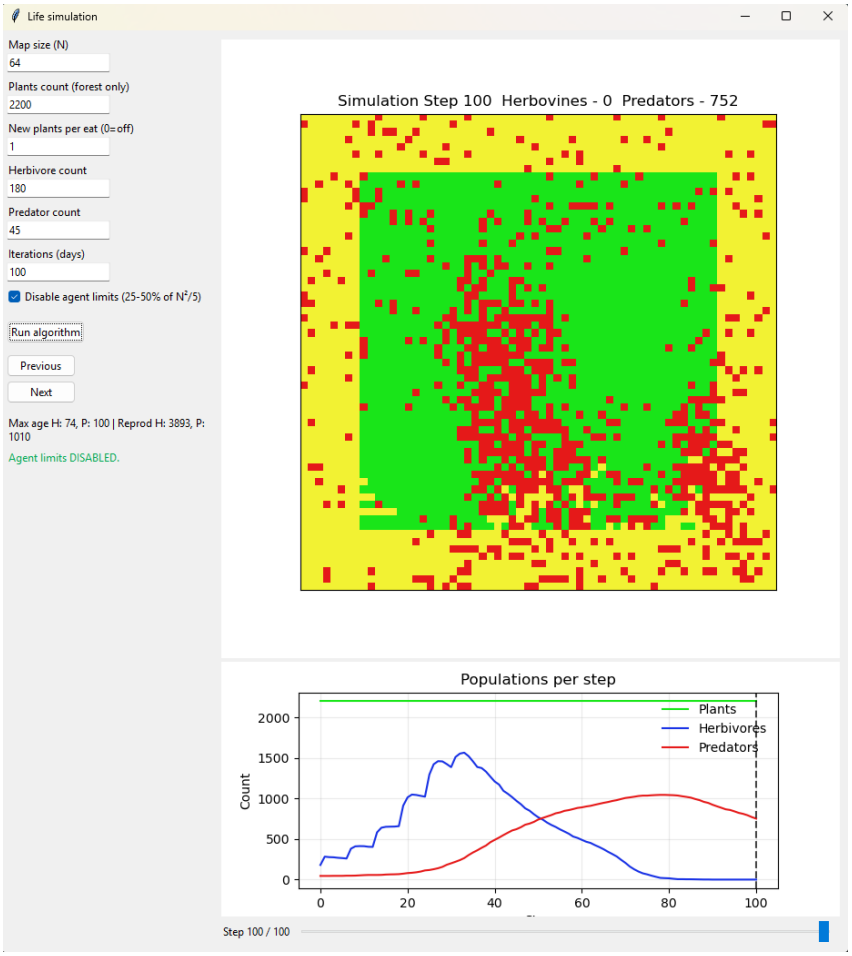
Iteration	H	P	Max age H	Max age P	Reprod H	Reprod P
1	6	58	60	60	84	6
2	9	63	60	60	94	8
3	3	58	60	60	90	9
4	4	62	60	60	94	12

5	3	67	60	60	99	7
Avg	5.00	61.60	60.00	60.00	92.20	8.40

За відсутності респавну рослин (NP=0) й стартовому балансі H=120 та P=120 травоїдні майже повністю зникають до 60-го кроку (H=5 у середньому). Хижаки тримаються довше, але їхня чисельність зменшується приблизно удвічі (P=61.6). Обидві групи мають Max age = 60, тобто частина особин доживає до кінця раунду. Розмноження у травоїдних було інтенсивним на початку (Reprod H=92.2), у хижаків помірним (Reprod P=8.4), однак без поповнення рослин популяція H неминуче вимирає, після чого падає й P. Поведінка відповідає очікуванням моделі при дефіциті ресурсу: спочатку – колапс жертви, далі – скорочення хижака. Рекомендація: зменшити початкове P або задати невеликий респавн рослин (NP=1–3), щоб уникнути вимирання H через брак їжі.

3) Стабільний ресурс:
N=64; P=2200; NP=1; H=180; P=45; I=100





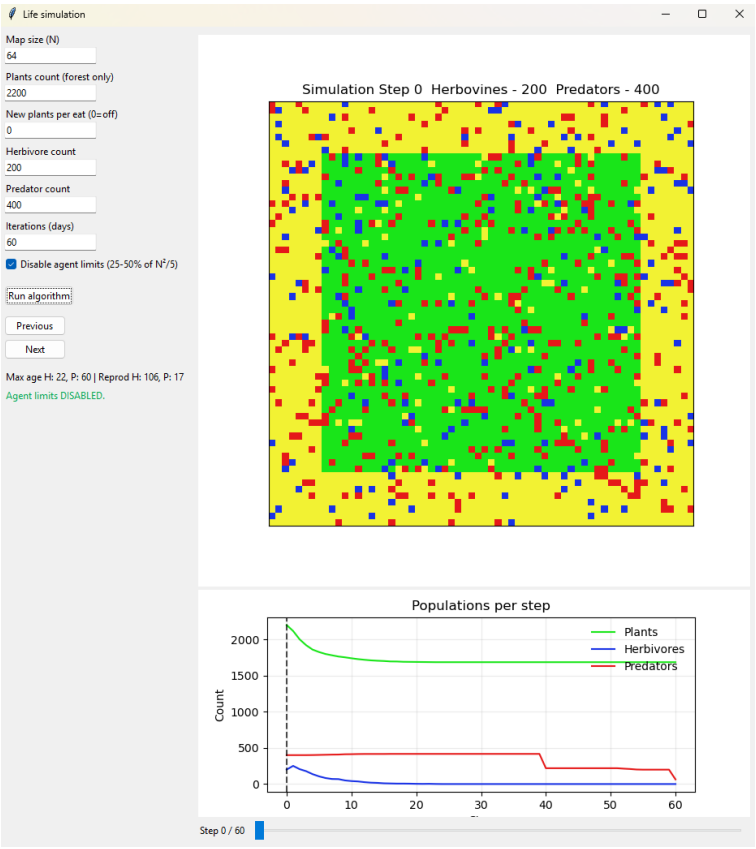
Iteration	H	P	Max age H	Max age P	Reprod H	Reprod P
1	0	752	74	100	3893	1010
2	0	863	67	100	4272	1108
3	0	788	81	100	4079	1070
4	0	774	70	100	4008	1045
5	0	800	71	100	4112	1071
Avg	0.00	795.40	72.60	100.00	4072.80	1060.80

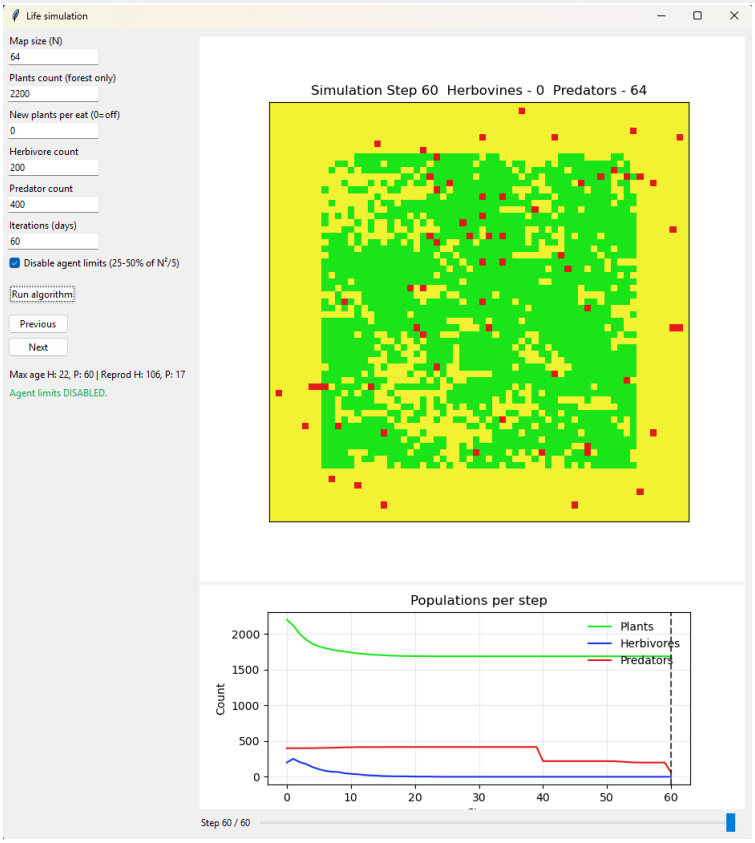
За умов NP=1, H=180, P=45 й великого запасу рослин система переходить у режим “хижаки-домінують”. На кінець 100-го кроку травоїдні зникають у всіх запусках (H=0), попри дуже інтенсивне розмноження (Reprod H=4 073); це означає, що тиск хижаків перекриває приріст H. Натомість чисельність хижаків зростає майже в 18 разів (до P=795 у середньому), а Max age P=100 свідчить, що значна частина доживає до кінця раунду. Max age H=73 показує довгі, але зрештою згасаючі агенти травоїдних.

Стабільний рослинний ресурс сам по собі не гарантує співіснування – за поточних параметрів P знищують H, після чого популяція P утримується на високому рівні. Рекомендації: зменшити стартове P або їхній приріст (менший E_GAIN_EAT_HERB / більший LOSS_PRED_PER_DAY), збільшити NP (напр., 2-3) чи знизити ймовірність атаки хижаками, щоб забезпечити стале співіснування.

4) Перенасичення хижаками:

N=64; P=2200; NP=0; H=200; P=400; I=60





Iteration	H	P	Max age H	Max age P	Reprod H	Reprod P
1	0	64	22	60	106	17
2	0	76	22	60	115	9
3	0	69	31	60	106	13
4	0	54	24	60	103	13
5	0	58	22	60	112	14
Avg	0.00	64.20	24.20	60.00	108.40	13.20

За NP=0, H=200, P=400 й 60 ітераціям популяція травоїдних повністю зникає (H=0 у всіх запусках) попри короткий стартовий сплеск розмноження (Reprod H=108.4). Хижаки після “початкового бенкету” різко скорочуються з 400 до P=64.2, але частина доживає до кінця (Max age P=60). Max age H=24.2 вказує, що окремі групи трималися недовго й були швидко знищені. Поведінка типова для перевищення тиску хижацтва при вичерпному ресурсі (NP=0): спершу – обвал жертви, далі – глибокий спад хижака. Рекомендації: зменшити стартове P (або E_GAIN_EAT_HERB/підвищити LOSS_PRED_PER_DAY), ввімкнути мінімальний респавн рослин (NP≥1) щоб запобігти швидкому колапсу.

Висновки: за результатом виконання лабораторної роботи створено інтелектуальну інформаційну системи на основі технології розвитку природних систем для моделювання процесів в економіці та живій природі, а також для моделювання взаємодії різноманітних об’єктів в конфліктних ситуаціях. Основні висновки за результатом дослідження:

- Отримано стійкі закономірності, узгоджені між собою на різних наборах параметрів:
  - Коректність базових механік. На малих прикладах без ресурсу (1-1: N=10, NP=0) травоїдні швидко зникають (H=>0, Max age H=2.2), що підтверджує роботу метаболізму/голоду. За наявності ресурсу й відсутності хижаків (1-2: N=16, P=120, NP=3, P=0) одна тварина відтворюється до 7.2 за 20 кроків (Max age H=20) – поріг відтворення працює коректно;
  - Дефіцит ресурсу (NP=0) – спершу колапс жертви, далі спад хижака. Для N=64, H=120, P=120, I=60 середні значення: H=5, P=61.6, Max age H=P=60, Reprod H=92.2 > Reprod P=8.4 лише на старті — коли трава вичерпується, травоїдні майже зникають, а хижаки тримаються інерційно, але теж зменшуються;
  - Стійкий (хоч і мінімальний) потік ресурсу різко змінює баланс: за NP=1 і високого запасу трави (N=64, P=2200, H=180, P=45, I=100) травоїдні зрештою зникають (H=0), натомість хижаки домінують (P=795.4, Max age P=100), попри надвисоке початкове розмноження H (Reprod H=4072.8). Стабільний ресурс без контролю хижака веде до “хижаки-домінують”;
  - Перенасичення хижаком при NP=0 дає подвійний колапс. N=64, P=2200, NP=0, H=200, P=400, I=60: H=0, P падає до 64.2, Max age P=60, Reprod P=13.2 – спочатку “переполювання” жертви, далі голод хижака.
- Динаміка відповідає логіці “ресурс => жертва => хижак”:
  - без респавну рослин жертва неминуче колапсує, а хижак тримається лише інерційно;
  - мінімальний респавн (NP>=1) без контролю P веде до домінації хижака;

- надлишок хижака при NP=0 дає подвійний колапс (спочатку H, потім P).

Вихідний код застосунку можна знайти за наступним посиланням на GitHub.

ДОДАТОК А

Вихідний код алгоритму

```
# -------------------- constants --------------------
class Obj(IntEnum):
    EMPTY = 0
    HERBIVORE = 2
    PREDATOR  = 3

DIRS = [(0,-1),(1,0),(0,1),(-1,0)]  # N,E,S,W

MAX_ENERGY_HERB = 100
MAX_ENERGY_PRED = 100
REPRO_THRESHOLD_HERB = int(0.9 * MAX_ENERGY_HERB)
REPRO_THRESHOLD_PRED = int(0.9 * MAX_ENERGY_PRED)

E_START_HERB = 80
E_START_PRED = 40

LOSS_HERB_PER_DAY = 2
LOSS_PRED_PER_DAY = 1

E_GAIN_EAT_HERB  = 20
E_GAIN_EAT_PLANT = 10

TURN_LEFT, TURN_RIGHT, FORWARD, EAT = 0, 1, 2, 3


# -------------------- tiny “brain” --------------------
@dataclass
class Brain:
    W: List[List[float]] = field(default_factory=lambda: [[random.uniform(-1,1) for _
in range(12)] for _ in range(4)])
    b: List[float]       = field(default_factory=lambda: [random.uniform(-0.5,0.5) for
_ in range(4)])

    def mutate(self, sigma: float = 0.05) -> 'Brain':
        Wm = [[w + random.gauss(0, sigma) for w in row] for row in self.W]
        bm = [bb + random.gauss(0, sigma) for bb in self.b]
        return Brain(Wm, bm)

    def forward(self, x12: List[float]) -> List[float]:
        y = []
        for r in range(4):
            s = self.b[r]
            wr = self.W[r]
            for i in range(12): s += wr[i] * x12[i]
            y.append(s)
        return y

# IMPORTANT: eq=False keeps identity equality & hash -> Agent is hashable for dict/set
@dataclass(eq=False)
class Agent:
    kind: Obj
    energy: int
    age: int
    generation: int
    x: int
    y: int
    gaze: int
    brain: Brain
    last_outputs: Tuple[float,float,float,float] = (0,0,0,0)
```

```
    idle: int = 0  # anti-stuck counter


# -------------------- world --------------------
class World:
    def __init__(self, N:int, plants:int, herbivores:int, predators:int,
                 seed:Optional[int]=None, per_eat_respawn:int=1,
enforce_caps:bool=True,
                 herbs_first:bool=True, panic_sidestep:bool=True,
anti_stuck:bool=True,
                 force_after:int=3):
        if seed is not None: random.seed(seed)
        self.N = N
        self.iteration = 0
        self.per_eat_respawn = max(0, int(per_eat_respawn))
        self.herbs_first = herbs_first
        self.panic_sidestep = panic_sidestep
        self.anti_stuck = anti_stuck
        self.force_after = max(1, force_after)

        self.anim_grid = [[Obj.EMPTY for _ in range(N)] for _ in range(N)]
        self.plant_grid = [[False for _ in range(N)] for _ in range(N)]
        self.animals: List[Agent] = []

        # centered 75% forest
        margin = max(1, int(round(N * 0.125)))
        self.forest = (margin, margin, N - margin - 1, N - margin - 1)

        # caps
        if enforce_caps:
            base = max(1, (N * N) // 5)
            lo, hi = int(0.25 * base), int(0.50 * base)
            herbivores = min(max(herbivores, lo), hi)
            predators  = min(max(predators,  lo), hi)

        self._place_plants(plants)
        self._spawn(Obj.HERBIVORE, herbivores)
        self._spawn(Obj.PREDATOR,  predators)

        self.stats = {
            "counts": [],
            "max_age": {Obj.HERBIVORE:0, Obj.PREDATOR:0},
            "reproductions": {Obj.HERBIVORE:0, Obj.PREDATOR:0},
        }
        self._log_counts(0)

    # ---- placement
    def _place_plants(self, count:int):
        x0,y0,x1,y1 = self.forest
        cells = [(x,y) for y in range(y0,y1+1) for x in range(x0,x1+1) if not
self.plant_grid[y][x]]
        random.shuffle(cells)
        for (x,y) in cells[:min(count, len(cells))]:
            self.plant_grid[y][x] = True

    def _spawn(self, kind:Obj, count:int):
        cells = [(x,y) for y in range(self.N) for x in range(self.N) if
self.anim_grid[y][x] == Obj.EMPTY]
        random.shuffle(cells)
        for (x,y) in cells[:min(count, len(cells))]:
            self.anim_grid[y][x] = kind
```

```
            e = E_START_HERB if kind==Obj.HERBIVORE else E_START_PRED
            self.animals.append(
                Agent(kind=kind, energy=e, age=0, generation=1,
                      x=x, y=y, gaze=random.randrange(4), brain=Brain())
            )

    def _forest_empties(self):
        x0,y0,x1,y1 = self.forest
        return [(x,y) for y in range(y0,y1+1) for x in range(x0,x1+1) if not
self.plant_grid[y][x]]

    def _respawn_plants(self, k:int):
        if k <= 0: return
        cells = self._forest_empties()
        if not cells: return
        random.shuffle(cells)
        for (x,y) in cells[:min(k, len(cells))]:
            self.plant_grid[y][x] = True

    # ---- helpers
    def _forward(self, a:Agent) -> Tuple[int,int]:
        dx,dy = DIRS[a.gaze]
        return (a.x + dx) % self.N, (a.y + dy) % self.N

    def _neighbors(self, x:int, y:int) -> List[Tuple[int,int]]:
        N=self.N
        return [((x+dx)%N,(y+dy)%N) for dx,dy in DIRS]

    def _sense12(self, a:Agent) -> List[float]:
        N=self.N
        def rot(dx,dy,g):
            if g==0: return dx,dy
            if g==1: return dy,-dx
            if g==2: return -dx,-dy
            return -dy,dx
        sec={"front":{"herb":0,"pred":0,"plant":0},
             "left":{"herb":0,"pred":0,"plant":0},
             "right":{"herb":0,"pred":0,"plant":0},
             "near":{"herb":0,"pred":0,"plant":0}}
        for dy in (-1,0,1):
            for dx in (-1,0,1):
                if dx==0 and dy==0: continue
                x=(a.x+dx)%N; y=(a.y+dy)%N
                k=self.anim_grid[y][x]
                if k==Obj.HERBIVORE: sec["near"]["herb"]+=1
                elif k==Obj.PREDATOR: sec["near"]["pred"]+=1
                if self.plant_grid[y][x]: sec["near"]["plant"]+=1
        for dy in range(-2,3):
            for dx in range(-2,3):
                if dx==0 and dy==0: continue
                rx,ry=rot(dx,dy,a.gaze)
                if abs(rx)<=2 and abs(ry)<=2:
                    if   ry < 0: name="front"
                    elif rx < 0: name="left"
                    elif rx > 0: name="right"
                    else: continue
                    x=(a.x+dx)%N; y=(a.y+dy)%N
                    k=self.anim_grid[y][x]
                    if k==Obj.HERBIVORE: sec[name]["herb"]+=1
                    elif k==Obj.PREDATOR: sec[name]["pred"]+=1
                    if self.plant_grid[y][x]: sec[name]["plant"]+=1
```

```
        out=[]; norm=1/12
        for key in ("front","left","right","near"):
            out.extend([sec[key]["herb"]*norm, sec[key]["pred"]*norm,
sec[key]["plant"]*norm])
        return out

    def _nearest(self, a:Agent, *, want_pred=False, want_herb=False,
want_plant=False)\
            -> Optional[Tuple[int,int,int]]:
        best=None; bestd=99
        for dy in range(-2,3):
            for dx in range(-2,3):
                if dx==0 and dy==0: continue
                x=(a.x+dx)%self.N; y=(a.y+dy)%self.N
                ok=False
                if want_pred and self.anim_grid[y][x]==Obj.PREDATOR: ok=True
                if want_herb and self.anim_grid[y][x]==Obj.HERBIVORE: ok=True
                if want_plant and self.plant_grid[y][x]: ok=True
                if not ok: continue
                d=abs(dx)+abs(dy)
                if d<bestd: bestd=d; best=(dx,dy,d)
        return best

    def _dir_toward(self, dx:int, dy:int) -> int:
        if abs(dx) >= abs(dy): return 1 if dx>0 else 3
        else: return 2 if dy>0 else 0

    def _cheb_dist(self, x1,y1,x2,y2):
        return max(abs(x1-x2), abs(y1-y2))

    def _plan_target(self, a:Agent) -> Tuple[int,int,bool]:
        N=self.N
        pounce = False
        pref_dir = a.gaze
        if a.kind == Obj.HERBIVORE:
            near_pred = self._nearest(a, want_pred=True)
            if near_pred is not None and near_pred[2] <= 2:
                dx,dy,_ = near_pred
                pref_dir = self._dir_toward(-dx, -dy)
            else:
                near_plant = self._nearest(a, want_plant=True)
                if near_plant is not None:
                    pref_dir = self._dir_toward(near_plant[0], near_plant[1])
        else:
            near_herb = self._nearest(a, want_herb=True)
            if near_herb is not None:
                pref_dir = self._dir_toward(near_herb[0], near_herb[1])
                fx, fy = (a.x + DIRS[pref_dir][0]) % N, (a.y + DIRS[pref_dir][1]) % N
                if self.anim_grid[fy][fx] == Obj.HERBIVORE:
                    pounce = True

        a.gaze = pref_dir
        fx, fy = self._forward(a)
        if self.anim_grid[fy][fx] == Obj.EMPTY:
            return fx, fy, pounce

        # panic sidestep
        left_dir  = (pref_dir - 1) % 4
        right_dir = (pref_dir + 1) % 4
        lx,ly = (a.x + DIRS[left_dir][0]) % N, (a.y + DIRS[left_dir][1]) % N
        rx,ry = (a.x + DIRS[right_dir][0]) % N, (a.y + DIRS[right_dir][1]) % N
```

```
    def safe(xx,yy): return self.anim_grid[yy][xx] == Obj.EMPTY
    if a.kind == Obj.HERBIVORE:
        pred = self._nearest(a, want_pred=True)
        if pred is not None:
            px,py = (a.x+pred[0])%N, (a.y+pred[1])%N
            dl = self._cheb_dist(lx,ly,px,py)
            dr = self._cheb_dist(rx,ry,px,py)
            for _,xx,yy in sorted([(dl,lx,ly),(dr,rx,ry)], reverse=True):
                if safe(xx,yy): return xx,yy,False
    else:
        herb = self._nearest(a, want_herb=True)
        if herb is not None:
            hx,hy = (a.x+herb[0])%N, (a.y+herb[1])%N
            dl = self._cheb_dist(lx,ly,hx,hy)
            dr = self._cheb_dist(rx,ry,hx,hy)
            for _,xx,yy in sorted([(dl,lx,ly),(dr,rx,ry)]):
                if safe(xx,yy): return xx,yy,False
    if self.anim_grid[ly][lx] == Obj.EMPTY: return lx,ly,False
    if self.anim_grid[ry][rx] == Obj.EMPTY: return rx,ry,False

    neigh = self._neighbors(a.x,a.y)
    empties = [(xx,yy) for (xx,yy) in neigh if self.anim_grid[yy][xx]==Obj.EMPTY]
    if len(empties) == 1:
        return empties[0][0], empties[0][1], False

    if a.kind == Obj.HERBIVORE:
        pred = self._nearest(a, want_pred=True)
        plant = self._nearest(a, want_plant=True)
        best=None; best_score=-1e9
        for xx,yy in empties:
            score = 0.0
            if pred is not None:
                px,py = (a.x+pred[0])%N,(a.y+pred[1])%N
                score += 10.0 * self._cheb_dist(xx,yy,px,py)
            if plant is not None:
                plx,ply = (a.x+plant[0])%N,(a.y+plant[1])%N
                score += -1.0 * self._cheb_dist(xx,yy,plx,ply)
            if score > best_score:
                best_score=score; best=(xx,yy)
        if best is not None:
            return best[0], best[1], False
    else:
        herb = self._nearest(a, want_herb=True)
        if herb is not None:
            hx,hy = (a.x+herb[0])%N,(a.y+herb[1])%N
            best=None; best_d=1e9
            for xx,yy in empties:
                d=self._cheb_dist(xx,yy,hx,hy)
                if d<best_d: best_d=d; best=(xx,yy)
            if best is not None:
                return best[0],best[1],False

    return a.x,a.y,False

# -------------------- one day --------------------
def step(self):
    self.iteration += 1

    herbs = [a for a in self.animals if a.kind==Obj.HERBIVORE]
    preds = [a for a in self.animals if a.kind==Obj.PREDATOR]
```

```
        move_order = random.sample(herbs, k=len(herbs)) + random.sample(preds,
k=len(preds))

        plans: Dict[Agent, Tuple[int,int,bool]] = {}
        for a in move_order:
            a.last_outputs = tuple(a.brain.forward(self._sense12(a)))
            plans[a] = self._plan_target(a)

        wishes: Dict[Tuple[int,int], List[Agent]] = {}
        for a,(tx,ty,_) in plans.items():
            wishes.setdefault((tx,ty), []).append(a)

        def rank(agent:Agent) -> int:
            return 1 if agent.kind==Obj.PREDATOR else 2

        moved=set()

        # swaps
        for a,(tx,ty,_) in list(plans.items()):
            if a in moved or (tx,ty)==(a.x,a.y): continue
            other_kind = self.anim_grid[ty][tx]
            if other_kind == Obj.EMPTY: continue
            b = next((o for o in self.animals if o.x==tx and o.y==ty and
o.kind==other_kind), None)
            if not b: continue
            btx,bty,_ = plans.get(b, (b.x,b.y,False))
            if (btx,bty) == (a.x,a.y):
                self.anim_grid[a.y][a.x], self.anim_grid[b.y][b.x] =
self.anim_grid[b.y][b.x], self.anim_grid[a.y][a.x]
                a.x,a.y, b.x,b.y = b.x,b.y, a.x,a.y
                a.idle = 0; b.idle = 0
                moved.add(a); moved.add(b)
                continue

        # contested cells
        for (cx,cy), claimers in wishes.items():
            movable = [ag for ag in claimers if (plans[ag][0],plans[ag][1]) !=
(ag.x,ag.y)]
            if not movable:
                continue
            if self.anim_grid[cy][cx] != Obj.EMPTY:
                continue
            by_rank={}
            for ag in movable:
                by_rank.setdefault(rank(ag), []).append(ag)
            top=min(by_rank.keys())
            winner = random.choice(by_rank[top]) if len(by_rank[top])>1 else
by_rank[top][0]
            if winner not in moved:
                self.anim_grid[winner.y][winner.x] = Obj.EMPTY
                winner.x,winner.y = cx,cy
                self.anim_grid[cy][cx] = winner.kind
                winner.idle = 0
                moved.add(winner)

        # remaining uncontested
        for a in move_order:
            if a in moved: continue
            tx,ty,_ = plans[a]
            if (tx,ty) == (a.x,a.y):
                a.idle += 1
```

```
                continue
            if self.anim_grid[ty][tx] == Obj.EMPTY:
                self.anim_grid[a.y][a.x] = Obj.EMPTY
                a.x,a.y = tx,ty
                self.anim_grid[ty][tx] = a.kind
                a.idle = 0
                moved.add(a)
            else:
                a.idle += 1

        # anti-stuck
        for a in self.animals:
            if a.idle >= 3:
                neigh = [(xx,yy) for (xx,yy) in self._neighbors(a.x,a.y) if
self.anim_grid[yy][xx]==Obj.EMPTY]
                if neigh:
                    if a.kind==Obj.HERBIVORE:
                        pred = self._nearest(a, want_pred=True)
                        if pred is not None:
                            px,py=(a.x+pred[0])%self.N,(a.y+pred[1])%self.N
                            neigh.sort(key=lambda p: -
self._cheb_dist(p[0],p[1],px,py))
                    else:
                        herb = self._nearest(a, want_herb=True)
                        if herb is not None:
                            hx,hy=(a.x+herb[0])%self.N,(a.y+herb[1])%self.N
                            neigh.sort(key=lambda p: self._cheb_dist(p[0],p[1],hx,hy))
                    nx,ny = neigh[0]
                    self.anim_grid[a.y][a.x] = Obj.EMPTY
                    a.x,a.y = nx,ny
                    self.anim_grid[ny][nx] = a.kind
                    a.idle = 0

        # pounce
        preds_now = [aa for aa in self.animals if aa.kind==Obj.PREDATOR]
        for a in random.sample(preds_now, k=len(preds_now)):
            nx,ny = self._forward(a)
            if self.anim_grid[ny][nx] == Obj.HERBIVORE:
                self.anim_grid[a.y][a.x] = Obj.EMPTY
                self.anim_grid[ny][nx]  = Obj.PREDATOR
                a.x,a.y = nx,ny
                a.energy = min(MAX_ENERGY_PRED, a.energy + E_GAIN_EAT_HERB)

        self.animals = [aa for aa in self.animals if self.anim_grid[aa.y][aa.x] ==
aa.kind]

        # herb eat
        herbs_now = [aa for aa in self.animals if aa.kind==Obj.HERBIVORE]
        for a in random.sample(herbs_now, k=len(herbs_now)):
            nx, ny = self._forward(a)
            if self.plant_grid[ny][nx]:
                self.plant_grid[ny][nx] = False
                a.energy = min(MAX_ENERGY_HERB, a.energy + E_GAIN_EAT_PLANT)
                self._respawn_plants(self.per_eat_respawn)

        # reproduce
        for a in random.sample(self.animals, k=len(self.animals)):
            thr = REPRO_THRESHOLD_HERB if a.kind==Obj.HERBIVORE else
REPRO_THRESHOLD_PRED
            if a.energy < thr:
                continue
```

```
            neigh = self._neighbors(a.x,a.y)
            random.shuffle(neigh)
            for nx,ny in neigh:
                if self.anim_grid[ny][nx] == Obj.EMPTY:
                    half = a.energy // 2
                    child_e = a.energy - half
                    a.energy = half
                    self.anim_grid[ny][nx] = a.kind
                    self.animals.append(
                        Agent(kind=a.kind, energy=child_e, age=0,
generation=a.generation+1,
                              x=nx, y=ny, gaze=random.randrange(4),
brain=a.brain.mutate(0.05))
                    )
                    self.stats["reproductions"][a.kind] += 1
                    break

        # metabolism
        survivors: List[Agent] = []
        for a in self.animals:
            a.age += 1
            if a.kind==Obj.HERBIVORE and a.age > self.stats["max_age"][Obj.HERBIVORE]:
                self.stats["max_age"][Obj.HERBIVORE] = a.age
            if a.kind==Obj.PREDATOR and a.age > self.stats["max_age"][Obj.PREDATOR]:
                self.stats["max_age"][Obj.PREDATOR] = a.age
            a.energy -= (LOSS_HERB_PER_DAY if a.kind==Obj.HERBIVORE else
LOSS_PRED_PER_DAY)
            if a.energy <= 0:
                self.anim_grid[a.y][a.x] = Obj.EMPTY
            else:
                survivors.append(a)
        self.animals = survivors

        self._log_counts(self.iteration)

    # -------------------- stats --------------------
    def _log_counts(self, i:int):
        plants = sum(1 for y in range(self.N) for x in range(self.N) if
self.plant_grid[y][x])
        herb   = sum(1 for a in self.animals if a.kind==Obj.HERBIVORE)
        pred   = sum(1 for a in self.animals if a.kind==Obj.PREDATOR)
        self.stats["counts"].append({"iter":i, "plants":plants, "herbivores":herb,
"predators":pred})
```

ДОДАТОК Б

Контрольні питання

1. Що таке штучне життя? З якою метою ведуться дослідження в галузі штучного життя?
   Штучне життя – це моделювання систем, що демонструють властивості живих: самопідтримку, адаптацію, еволюцію, відтворення. Дослідження дають розуміння екології та еволюції, допомагають проектувати автономних агентів, алгоритми керування, створювати нові підходи до робототехніки та біоінженерії.
2. Яким чином пов’язані штучне життя та синтетична наука про поведінку?
   Обидві галузі будують штучні (синтетичні) системи й перевіряють, чи виникає потрібна поведінка з простих правил на основі взаємодії з навколишнім середовищем. Штучне життя – підхід для перевірки гіпотез про поведінку та еволюцію агентів, пристосування до навколишнього середовища.
3. В чому полягає сутність моделювання харчових ланцюгів?
   Це симуляція взаємодій “ресурс => жертва => хижак” з отриманням енергії під час поїдання, обмеженнями на розмноження/смертність і просторовою конкуренцією.
4. Проаналізуйте параметри, які визначаються при моделюванні харчових ланцюгів.
   При моделюванні харчових ланцюгів враховуються наступні параметри:
   - Розмір середовища (N);
   - Кількість енергії, яку мають організми;
   - Кількість енергії, яку агенти отримують або втрачають за певні дії;
   - Взаємодії між агентами та рівні харчового ланцюга;
   - Рівні харчового ланцюга;
   - Смертність;
   - Відтворення;
   - Додаткові обмеження за потреби.
5. Дайте визначення понять: область близькості агенту, фронт агенту.
   Область близькості – область навколо агента, в межах якої він може спостерігати або взаємодіяти з іншими об'єктами чи агентами.
   Фронт агента – орієнтація (N/E/S/W), в якому агент рухається або орієнтований. Це може бути вектор, що вказує на рух агента або на його напрямок спостереження.
6. З якою метою використовуються сенсори та активатори?
   Сенсори надають агенту вхідну інформацію для “сприйняття” середовища та корекції поведінки залежно від зовнішніх умов (у нашому випадку кількість H/P/рослин у секторах (front/left/right/near)).
   Активатори виконують фізичні або віртуальні дії агента (повернутися, рушити вперед на 1 клітинку, атакувати у хижака, з’їсти, розмножитися).
7. Які математичні засоби можуть бути використані для моделювання мозку агента?
   Скінченні автомати, еволюційні алгоритми, Q-навчання/RL (у нас – лінійна політика).
8. Які дії може виконувати агент?
   Агент здатен виконувати наступні дії:

- Переміщення в межах мапи (повернути ліворуч/праворуч/зміна фронту, крок уперед на 1 клітинку);
- Поглинання їжі (атака хижака або травоїдні і поглинання рослини);
- Взаємодія з іншими агентами (як із іншого виду, так і свого);
- Розмноження за умов порогу енергії.

9. Що таке енергія та метаболізм?
   Енергія – це ресурс, необхідний агенту для виконання дій (рух, відтворення). Вона може бути отримана через споживання їжі чи інших ресурсів. При закінченні енергії – агент “помирає”.
   Метаболізм – це процес обміну енергії в агенті, включаючи її витрати та відновлення. Метаболізм також може включати споживання їжі та витрати енергії для підтримки життєдіяльності.
10. При виконанні якої умови агент допускається до відтворення?
    Коли енергія агента задовольняє умові для відтворення (у нас – >= 90%), є вільна сусідня клітинка; енергія ділиться між батьком і дитиною, “мозок” дитини може отримати мутацію.
11. Якими способами агент може загинути?
    - Голод (енергія <= 0);
    - Взаємодія з іншим агентом (будучи з’їденим хижаком (для травоїдних)).