🌍 Hackathon : Simulation d'Écosystème en Python

📋 Introduction

Bienvenue au défi de simulation d'écosystème ! Vous allez créer un monde virtuel où des moutons, des loups et de l'herbe interagissent dans un environnement dynamique. Ce projet vous permettra d'explorer les concepts de programmation orientée objet, de simulation et de visualisation de données.

Durée : 1 journée (6-8 heures de développement)

🎯 Objectif Principal

Développer une simulation d'écosystème en Python où :

- Des **moutons** broutent l'herbe pour survivre
- Des **loups** chassent les moutons pour se nourrir
- L'**herbe** repousse aléatoirement sur la grille
- Les populations évoluent dynamiquement selon les règles de l'écosystème

---

🔧 Spécifications Techniques

1. Système de Grille

- Grille carrée de taille **n × n** (recommandé : 20 ≤ n ≤ 50)
- Chaque case peut contenir :
  - De l'herbe (ou sol vide)
  - **Un seul animal maximum** (mouton OU loup, pas les deux)
  - Herbe + un animal simultanément

Comportement aux bords : La grille est fermée (les bords sont des limites infranchissables).

Note sur l'adjacence : "Adjacence" ou "distance = 1" signifie les **4 cases orthogonales** (haut, bas, gauche, droite), **sans les diagonales**.

2. Entités de l'Écosystème

🐑 **Moutons (Sheep)**

Propriétés minimales :

- Position (x, y)
- Niveau d'énergie (diminue à chaque tour)
- Âge

Comportements :

- **Déplacement** : Se déplacent aléatoirement vers une case adjacente libre
    - Si de l'herbe est détectée à distance = 1, se déplacent automatiquement vers cette case au prochain tour
- **Alimentation** : Doivent être **sur la même case** que l'herbe pour la manger et gagner de l'énergie (+10-20 points)
- **Reproduction** : Se reproduisent de manière **autonome** si énergie > seuil (ex: 50)
    - Pas besoin d'un autre mouton pour se reproduire
    - Coûte 20 d'énergie au parent
    - Crée un nouveau mouton sur une case adjacente libre avec l'énergie initiale (20)
- **Mort** : Meurent si énergie ≤ 0 ou âge > limite

🐺 **Loups (Wolves)**

Propriétés minimales :

- Position (x, y)
- Niveau d'énergie (diminue plus rapidement que les moutons)
- Âge

Comportements :

- **Déplacement** : Se déplacent aléatoirement vers une case adjacente libre
    - Si un mouton est détecté à distance = 1, se déplacent automatiquement vers cette case au prochain tour
- **Alimentation** : Mangent les moutons **en adjacence** (cases voisines, distance = 1) pour gagner de l'énergie (+30-40 points)
    - Le mouton mangé disparaît de la grille
- **Reproduction** : Se reproduisent de manière **autonome** si énergie > seuil (ex: 80)
    - Pas besoin d'un autre loup pour se reproduire
    - Coûte 20 d'énergie au parent
    - Crée un nouveau loup sur une case adjacente libre avec l'énergie initiale (40)
- **Mort** : Meurent si énergie ≤ 0 ou âge > limite

🌱 **Herbe (Grass)**

Propriétés :

- État : présente ou absente
- Temps de repousse (si mangée)

Comportements :

- **Nouvelle herbe** : Apparaît aléatoirement sur les cases vides sans herbe (probabilité : 5-10% par tour)
- **Herbe mangée** : Disparaît quand consommée par un mouton
- **Repousse** : L'herbe mangée repousse sur la même case après un délai fixe (ex: 7 tours)

3. Règles de Simulation

Cycle de simulation (1 tour) :

1. Incrémenter l'âge de tous les animaux (+1)
2. Mise à jour de l'herbe (repousse de l'herbe mangée + apparition aléatoire)
3. Phase Moutons : détection, déplacement, alimentation, perte d'énergie (-1)
4. Phase Loups : détection, déplacement, chasse, perte d'énergie (-2)
5. Vérification des morts (énergie ≤ 0 ou âge > limite)
6. Reproduction (si conditions remplies)
7. Affichage de l'état actuel
8. Vérification des conditions d'arrêt

Paramètres suggérés :

```
# Configuration initiale
GRID_SIZE = 30
INITIAL_SHEEP = 50
INITIAL_WOLVES = 10
INITIAL_GRASS_COVERAGE = 0.3  # 30% de la grille

# Énergie
SHEEP_INITIAL_ENERGY = 20
WOLF_INITIAL_ENERGY = 40
SHEEP_ENERGY_FROM_GRASS = 15
WOLF_ENERGY_FROM_SHEEP = 35
SHEEP_ENERGY_LOSS_PER_TURN = 1
WOLF_ENERGY_LOSS_PER_TURN = 2  # Les loups perdent plus d'énergie

# Reproduction
SHEEP_REPRODUCTION_THRESHOLD = 50
WOLF_REPRODUCTION_THRESHOLD = 80
REPRODUCTION_ENERGY_COST = 20

# Âge
SHEEP_MAX_AGE = 50  # Tours avant mort naturelle
WOLF_MAX_AGE = 40

# Herbe
GRASS_GROWTH_PROBABILITY = 0.08
GRASS_REGROWTH_TIME = 7

# Simulation
MAX_TURNS = 500  # Nombre maximum de tours
```

Placement initial :

- Herbe : 30% des cases aléatoirement
- Moutons : 50 moutons sur cases aléatoires (peuvent être sur de l'herbe), énergie = 20, âge = 0
- Loups : 10 loups sur cases aléatoires libres (sans autre animal), énergie = 40, âge = 0

4. Conditions d'Arrêt

La simulation s'arrête si :

1. Nombre maximum de tours atteint (500 par défaut)
2. Extinction totale (tous les animaux sont morts)
3. Arrêt manuel (Ctrl+C)

5. Gestion des Conflits

Conflit de déplacement : Si plusieurs animaux veulent la même case, le premier traité (ordre aléatoire) l'obtient.

Conflit de chasse : Si plusieurs loups ciblent le même mouton, seul le premier traité le mange.

📦 Livrables Attendus

Obligatoire

1. **Code source Python** :
   - Fichier principal (`main.py` ou `simulation.py`)
   - Classes pour les entités (Sheep, Wolf, Grass, Grid)
   - Code commenté et structuré
2. **Visualisation dans le terminal** :
   - Affichage textuel de la grille avec des caractères ASCII
   - Représentation claire des différentes entités (moutons, loups, herbe)
   - Mise à jour en temps réel de l'affichage

   Symboles suggérés :
   - `S` ou 🐑 = Mouton (Sheep)
   - `W` ou 🐺 = Loup (Wolf)
   - `#` ou 🌱 = Herbe (Grass)
   - `.` ou ⬜ = Case vide
3. **Documentation** :
   - Fichier `README.md` expliquant :
     - Comment lancer la simulation
     - Paramètres configurables
     - Règles implémentées

Optionnel

- Fichier de configuration (JSON/YAML)
- Tests unitaires

- Affichage coloré dans le terminal (avec ANSI codes)
- Statistiques détaillées après chaque tour

📊 Critères d'Évaluation

Critère	Points	Description
Fonctionnalité	40 pts	Simulation fonctionnelle avec toutes les entités
Qualité du code	20 pts	Structure, lisibilité, commentaires
Visualisation	15 pts	Clarté de l'affichage de l'écosystème
Créativité	15 pts	Fonctionnalités bonus et originalité
Documentation	10 pts	README clair et complet
TOTAL	100 pts	

Note : Les points bonus s'ajoutent aux points de base (pas de limite maximale). Le classement final est basé sur le score total.

🚀 Fonctionnalités Bonus

Implémentez ces fonctionnalités pour obtenir des points supplémentaires :

Niveau 1 (+10 pts chacune)

- ✨ **Statistiques en temps réel** : Affichage du nombre de chaque espèce et statistiques détaillées
- 🎨 **Affichage coloré** : Utilisation de codes ANSI pour colorer le terminal
- 📊 **Tableau de bord** : Panneau d'informations avec historique des populations
- ⚙️ **Fichier de configuration** : Paramètres modifiables sans toucher au code
- 🖥️ **Affichage fixe dans le terminal** : Mise à jour de la grille sur place (sans scroll) en utilisant des séquences d'échappement ANSI ou des bibliothèques comme `curses` pour effacer et redessiner la grille au même endroit

Niveau 2 (+15 pts chacune)

- 🧠 **IA pour les loups** : Déplacement intelligent vers les moutons
- 👥 **Comportement de groupe** : Les moutons fuient les loups
- 🌊 **Obstacles** : Rivières, montagnes (cases infranchissables)
- 💾 **Sauvegarde/Chargement** : Sauvegarder l'état de la simulation

Niveau 3 (+20 pts chacune)

- ⌨️ **Contrôles interactifs** : Pause, vitesse, ajout manuel d'entités via clavier
- 🔬 **Analyse écologique** : Calcul de la biodiversité, équilibre de l'écosystème
- 🌡️ **Saisons** : Variations saisonnières affectant la croissance de l'herbe

- 🧬 **Génétique** : Traits héréditaires (vitesse, vision, etc.)

Niveau 4 (+25 pts)

- 📈 **Graphiques d'évolution** : Utilisation de matplotlib pour afficher l'évolution des populations (moutons, loups, herbe) au fil du temps à la fin de la simulation. Le graphique doit montrer clairement les tendances et interactions entre les différentes espèces.

🛠️ Contraintes Techniques

- **Langage** : Python 3.8 ou supérieur
- **Bibliothèques autorisées** :
  - Bibliothèque standard Python (random, time, os, sys, etc.)
  - numpy (optionnel, pour gestion de grille)
  - colorama ou termcolor (optionnel, pour couleurs dans le terminal)
  - matplotlib (optionnel, pour graphiques d'évolution des populations)
- **Interdictions** :
  - Pas de frameworks de simulation existants
  - Pas de code généré par IA (ChatGPT, Copilot, etc.)
  - Le code doit être original

📝 Format de Soumission

Structure du Projet

```
nom_equipe_ecosysteme/
├── README.md
├── main.py (ou simulation.py)
├── entities.py (classes Sheep, Wolf, Grass)
├── grid.py (classe Grid)
├── config.json (optionnel)
├── requirements.txt (si bibliothèques externes)
```

Modalités de Soumission

- **Deadline :** [DATE ET HEURE À PRÉCISER]
- **Format :** Archive ZIP nommée `nom_equipe_ecosysteme.zip`
- **Méthode :** [À PRÉCISER - email, plateforme, dépôt Git, etc.]

Vérifications Avant Soumission

- ☐ Le code s'exécute sans erreur sur Python 3.8+
- ☐ Le README explique clairement comment lancer la simulation
- ☐ Les paramètres sont configurables
- ☐ L'affichage est clair et lisible

🏆 Bon Courage !

N'oubliez pas : l'objectif est d'apprendre et de s'amuser ! Ne vous découragez pas si tout ne fonctionne pas parfaitement. Une simulation simple mais fonctionnelle vaut mieux qu'un projet ambitieux inachevé.

Questions ? N'hésitez pas à demander de l'aide aux organisateurs.