MODÉLISATION INFORMATIQUE D'UN SYSTÈME PROIES-PRÉDATEURS : DU RÉEL AU SIMULÉ

CHAABNA Hakim

Lycée Saint-Rémi, TIPE 2025

INTRODUCTION

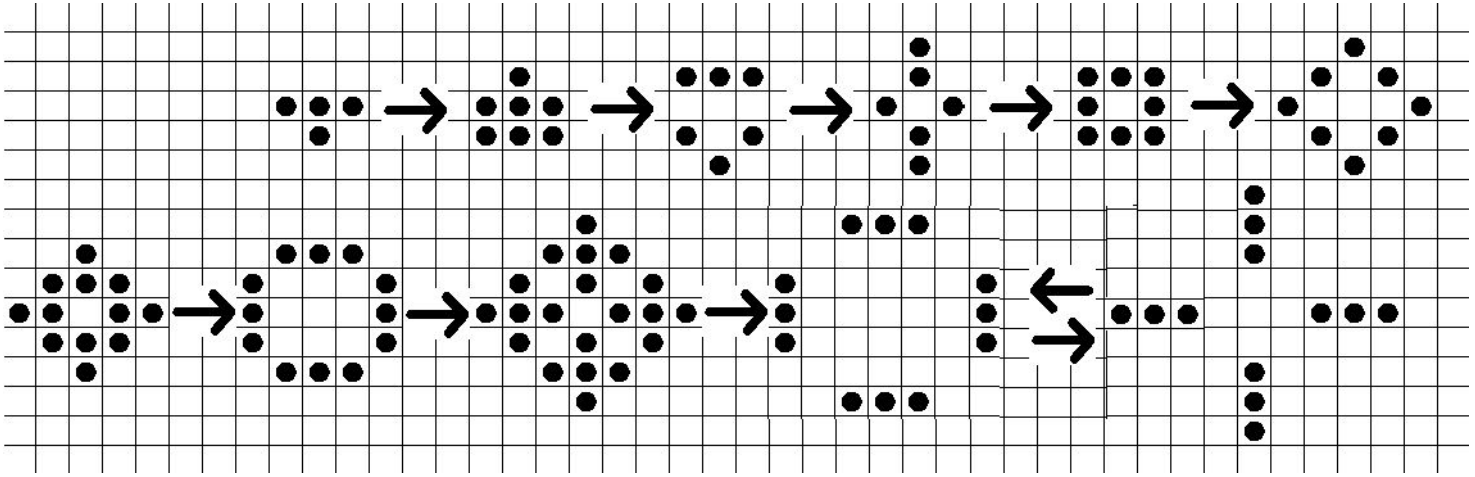


Figure 1

À partir de règles simples, un comportement complexe peut émerger…

PROBLÉMATIQUE

Dans quelle mesure peut-on reproduire les cycles d'évolution d'un système proies-prédateurs à l'aide d'une simulation informatique ?

PLAN

- Contexte scientifique : le cas de l'Isle Royale
- Modélisation des données réelles
- Création d'un modèle informatique simulé
- Résultats de la simulation
- Comparaison théorie/pratique
- Limites du modèle
- Conclusion

CONTEXTE SCIENTIFIQUE : LE CAS DE L'ISLE ROYALE

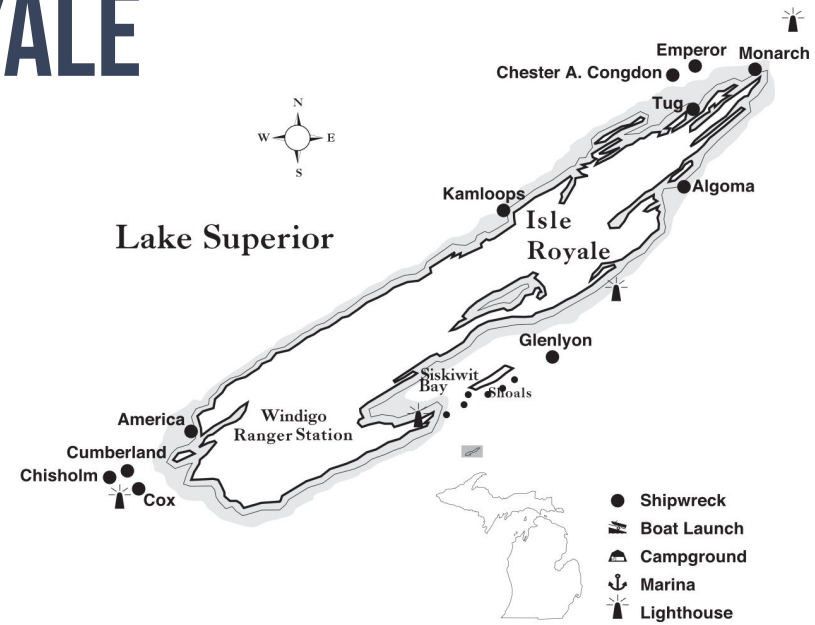


Figure 2

- Étude sur l'Isle Royale (Michigan) depuis 1958.
  - Observation des populations de loups 🐺 et d'orignaux 🦌.
- Base réelle pour une modélisation informatique.
  - Écosystème isolé = conditions idéales.
- Mise en évidence de cycles proies-prédateurs.

MODÉLISATION DES DONNÉES RÉELLES

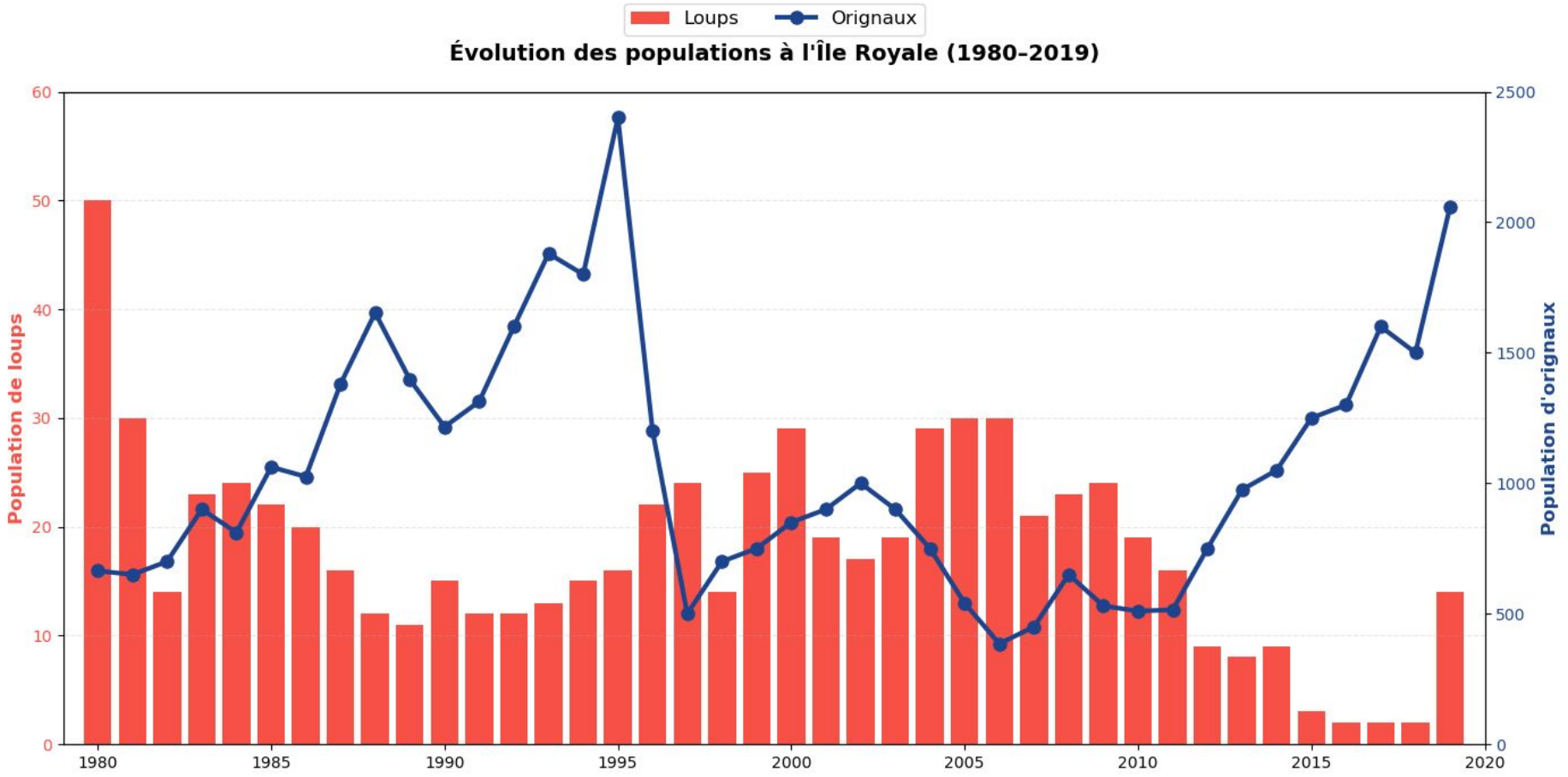
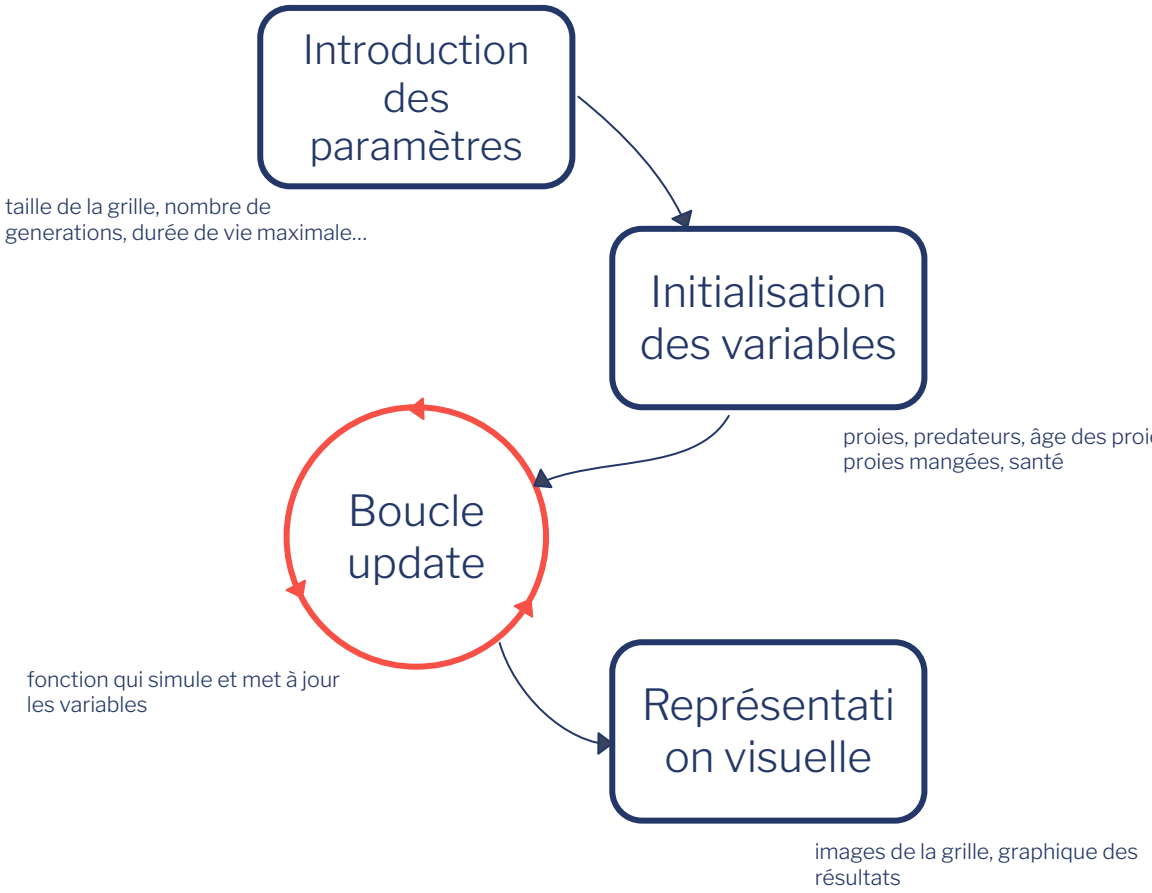


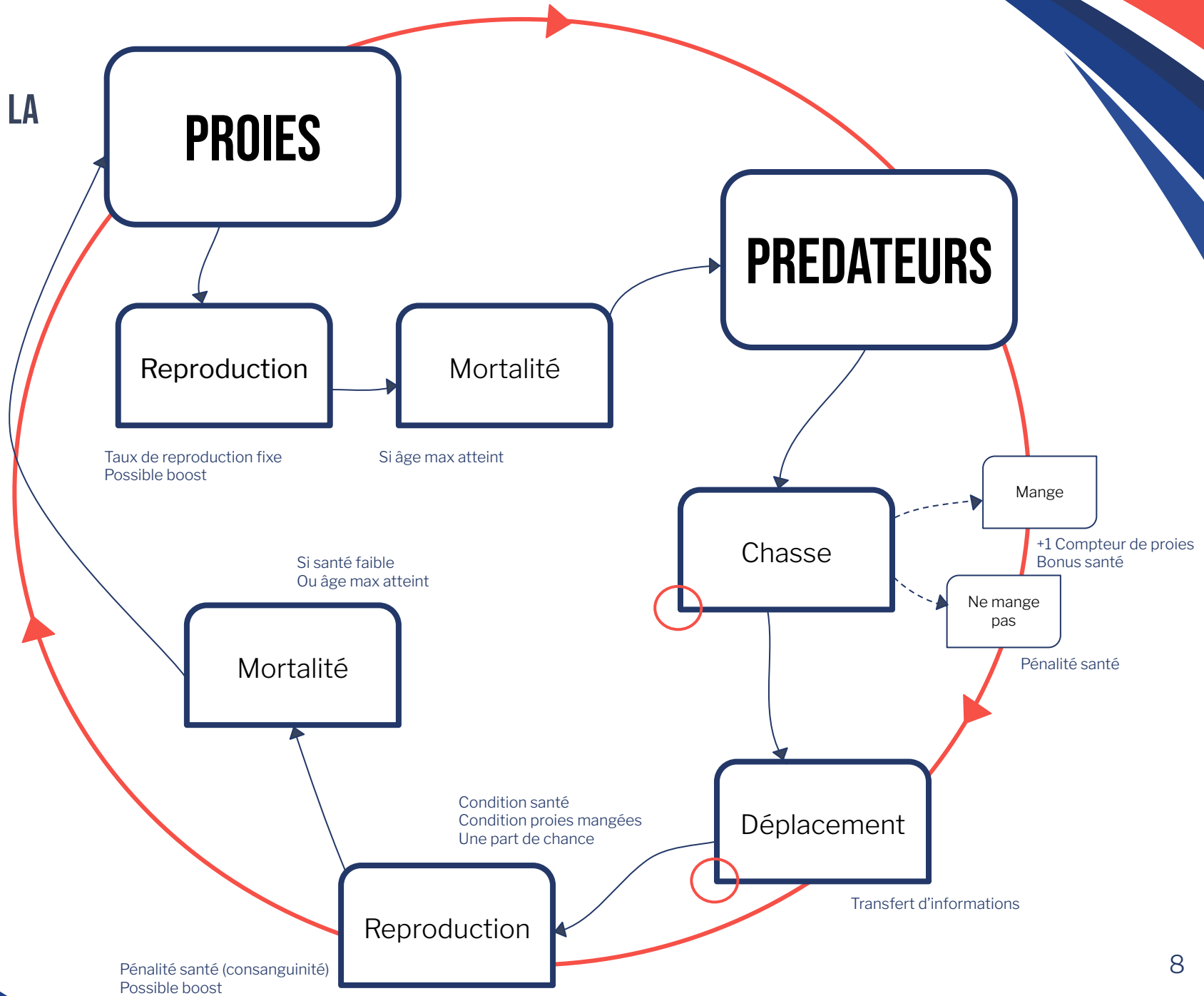
Figure 3

CRÉATION D'UN MODÈLE INFORMATIQUE SIMULÉ

STRUCTURE DU PROGRAMME



ZOOM SUR LA BOUCLE UPDATE

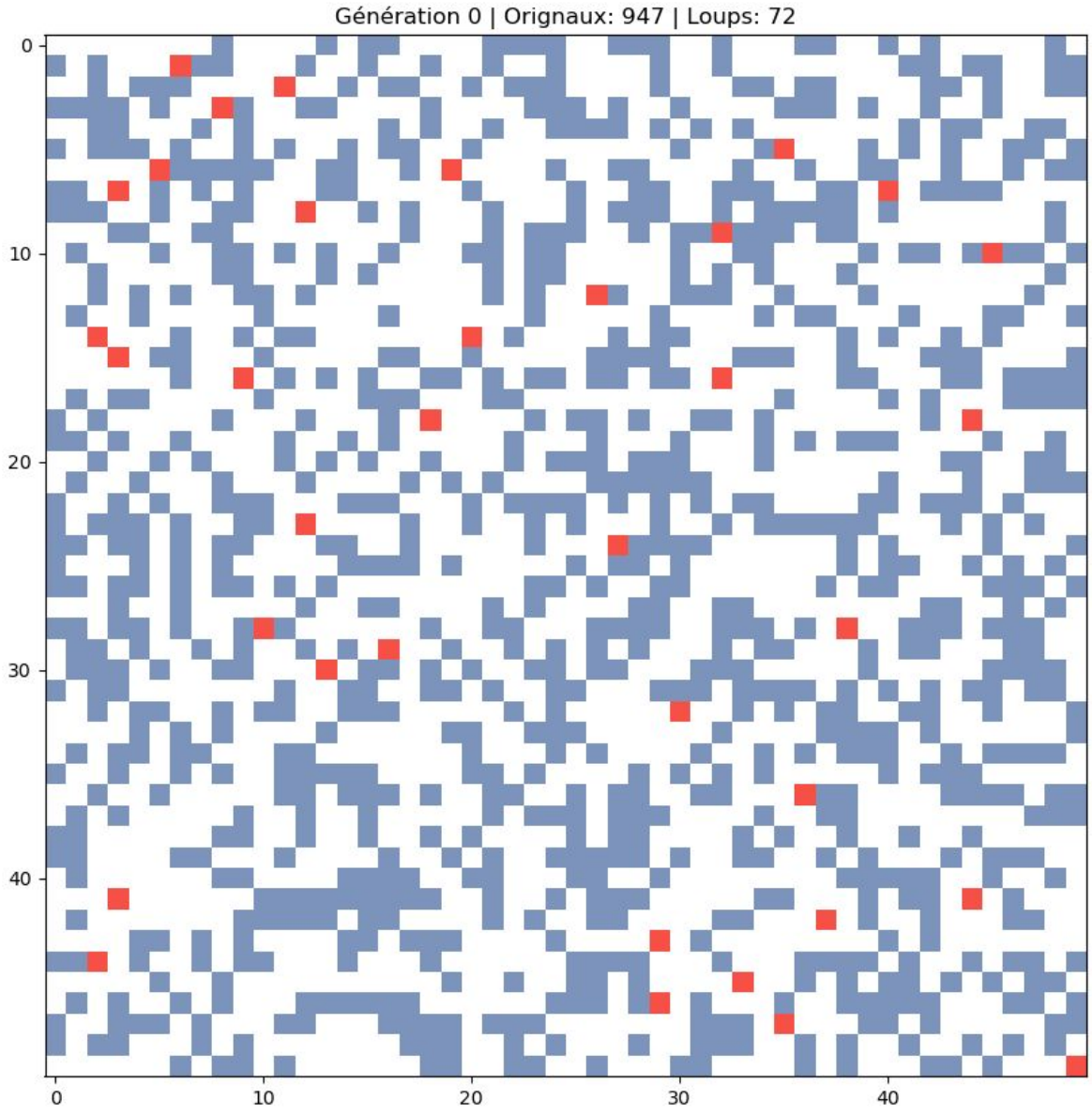


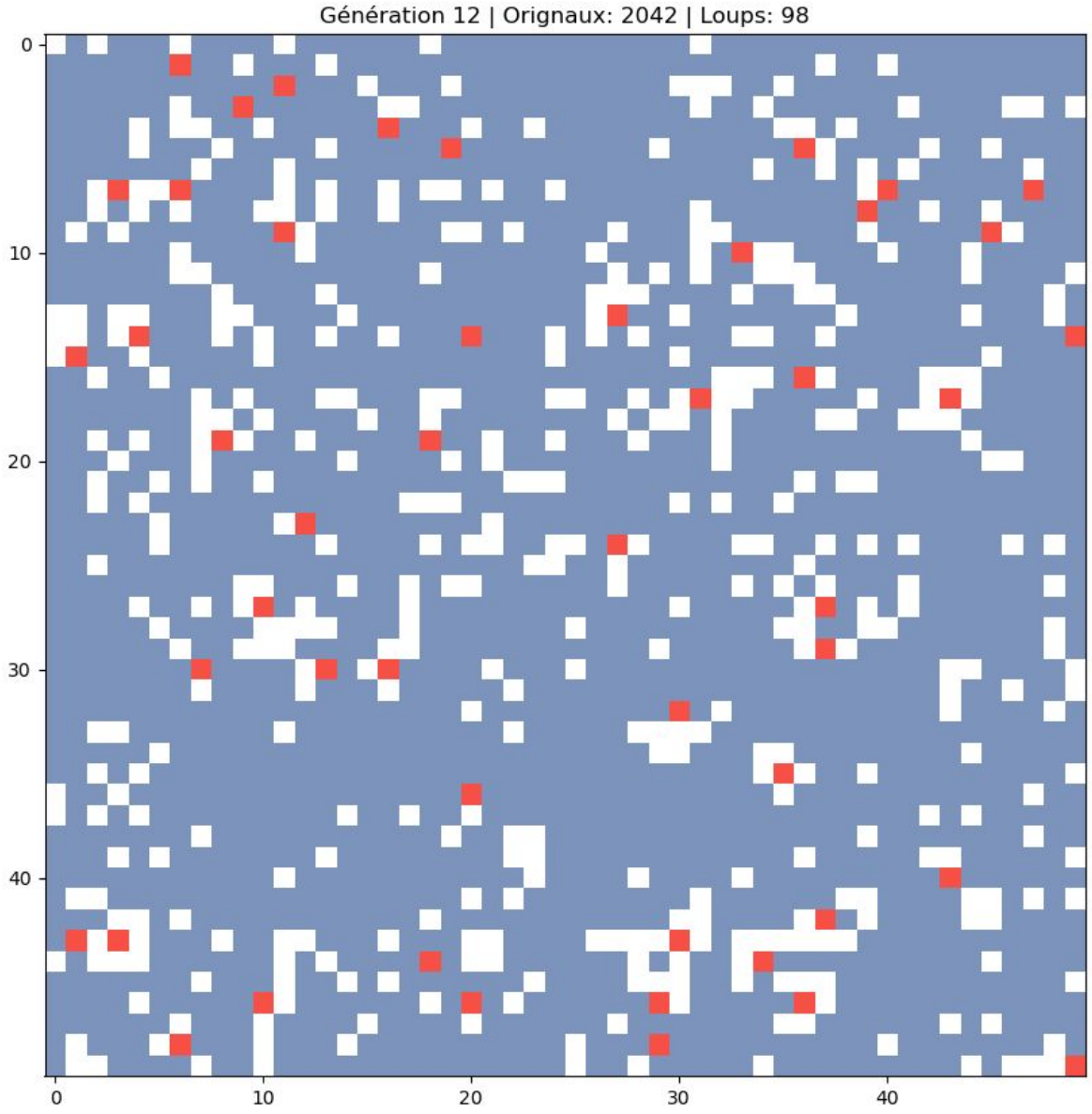
ZOOM SUR LE COMPORTEMENT DES PRÉDATEURS

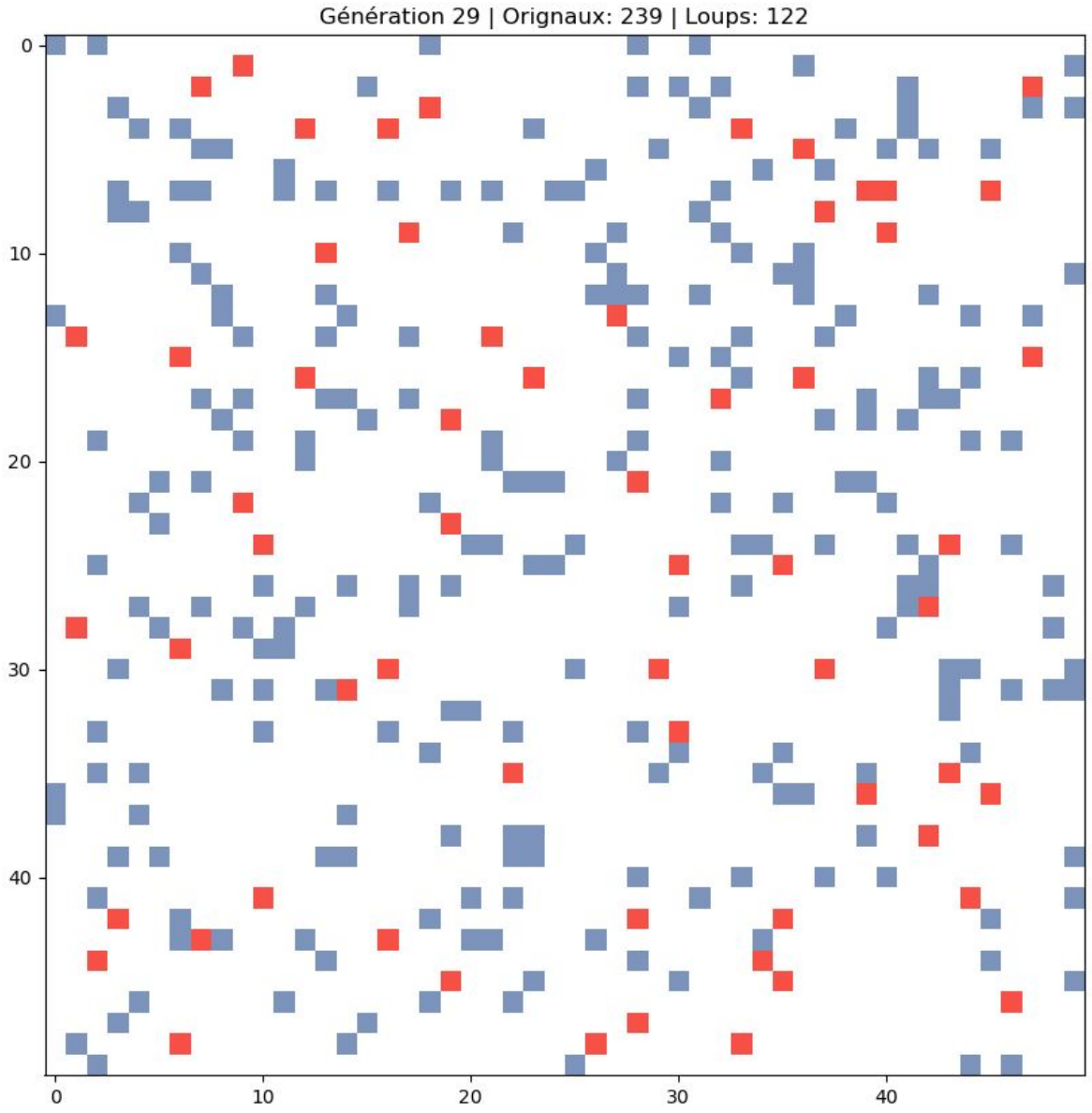
```
# Comportement des prédateurs
elif predateurs[i, j] == 2:

    nouv_age[i, j] += 1  # Vieillissement

    # Liste les voisins
    voisins = [(i+di, j+dj) for di in [-1, 0, 1] for dj in [-1, 0, 1]
               if 0 <= i+di < size and 0 <= j+dj < size and (di != 0 or dj != 0)]
    np.random.shuffle(voisins)
    deplacement = False

    # Phase 1 : Chasse
    for ni, nj in voisins:
        if nouv_proies[ni, nj] == 1:
            nouv_proies[ni, nj] = 0  # Mange la proie
            nouv_proies_mangees[i, j] += 1
            nouv_sante[i, j] = nouv_sante[i, j] + 0.1  # Améliore la santé
            deplacement = True
            break

    # Phase 2 : Déplacement
    if not deplacement:
        nouv_sante[i, j]= nouv_sante[i, j] -0.25  # Pénalité santé si ne mange pas
        for ni, nj in voisins:
            if nouv_predateurs[ni, nj] == 0:
                nouv_predateurs[i, j] = 0
                nouv_predateurs[ni, nj] = 2
                nouv_proies_mangees[ni, nj] = nouv_proies_mangees[i, j]
                nouv_age[ni, nj] = nouv_age[i, j]
                nouv_sante[ni, nj] = nouv_sante[i, j]
                break
```

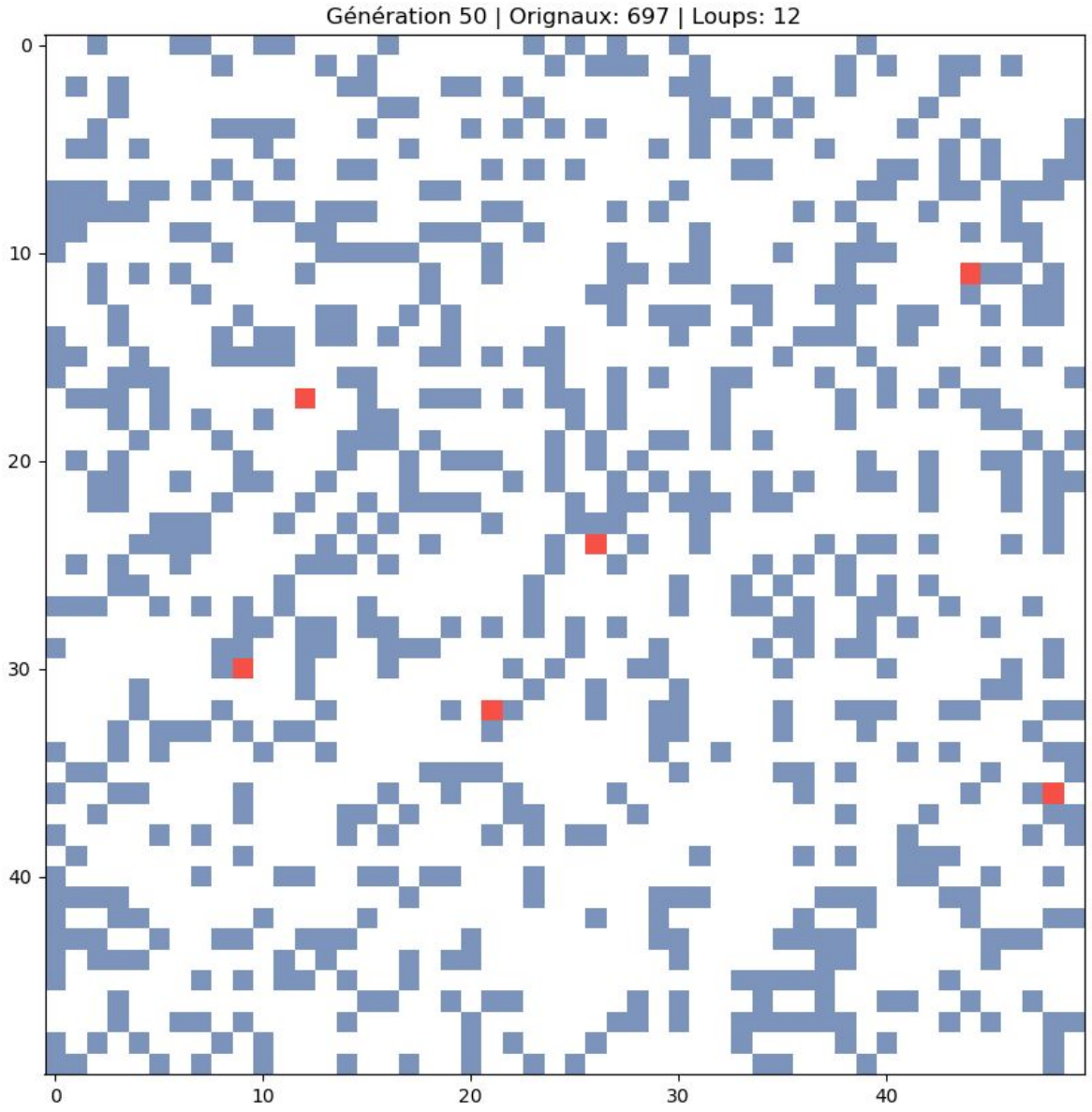
```python
if np.sum(predateurs) < 25:  # Seuil critique de loups
    nouv_sante[i, j] = max(1.2, nouv_sante[i, j] + 0.3)  # Bonus santé
    nouv_seuil_reproduction_loup = 4  # Réduction du seuil nécessaire

# Phase 3 : Reproduction
if (nouv_proies_mangees[i, j] >= nouv_seuil_reproduction_loup and
    nouv_sante[i, j] > 1.2 and
    np.random.rand() < 0.4):
    ni, nj = np.random.randint(0, size, 2)
    if nouv_predateurs[ni, nj] == 0:
        nouv_predateurs[ni, nj] = 2
        nouv_proies_mangees[ni, nj] = 0  # Proies mangées par le nouveau loup
        nouv_age[ni, nj] = 0
        nouv_sante[ni, nj] = 0.8  # Santé initiale réduite (consanguinité)
        nouv_proies_mangees[i, j] = 0  # Reset parent

# Phase 4 : Mortalité
if (nouv_age[i, j] > duree_vie_max1 or
    nouv_sante[i, j] <= 0.4):
    nouv_predateurs[i, j] = 0
```

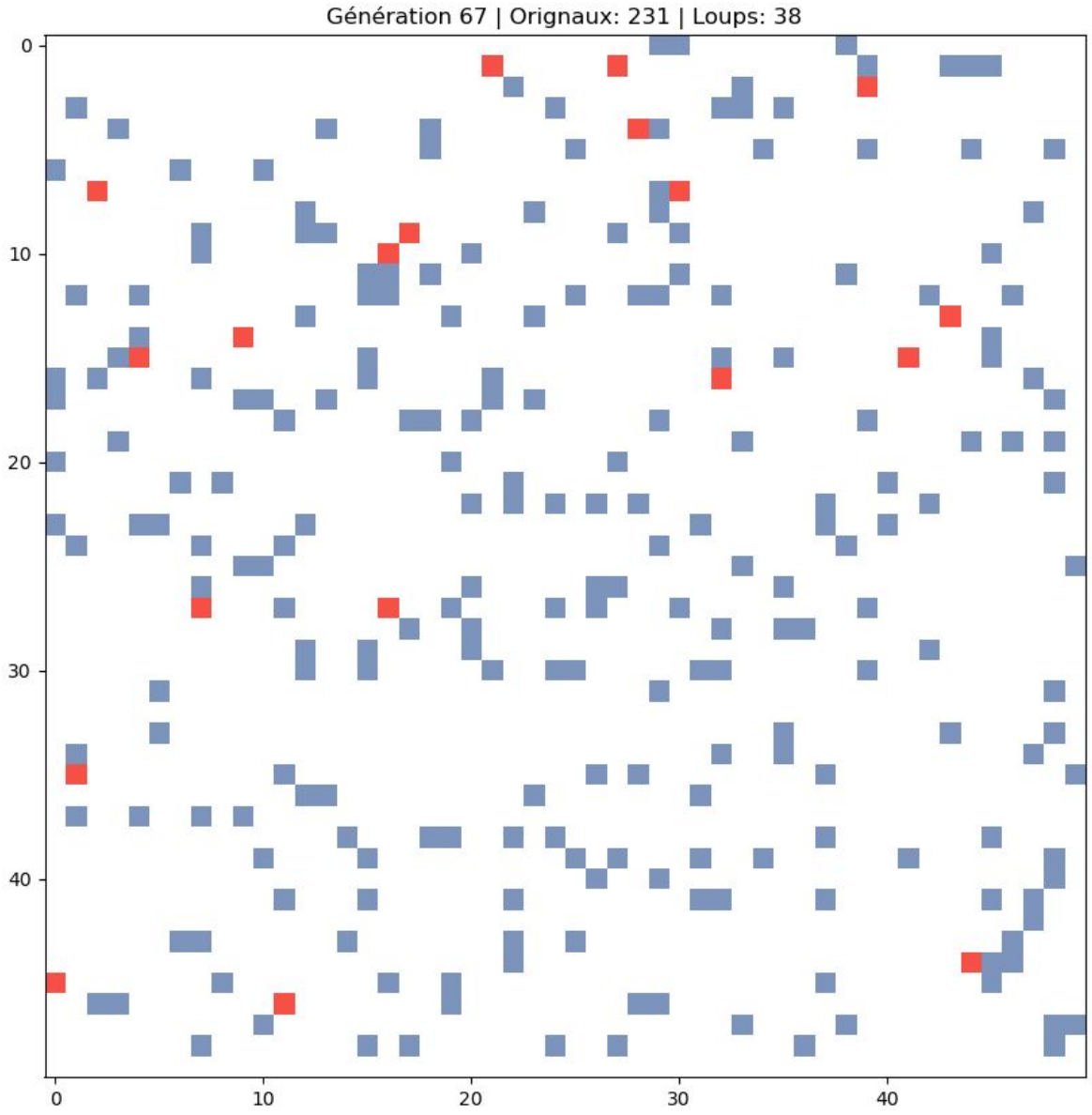
RÉSULTATS DE LA SIMULATION

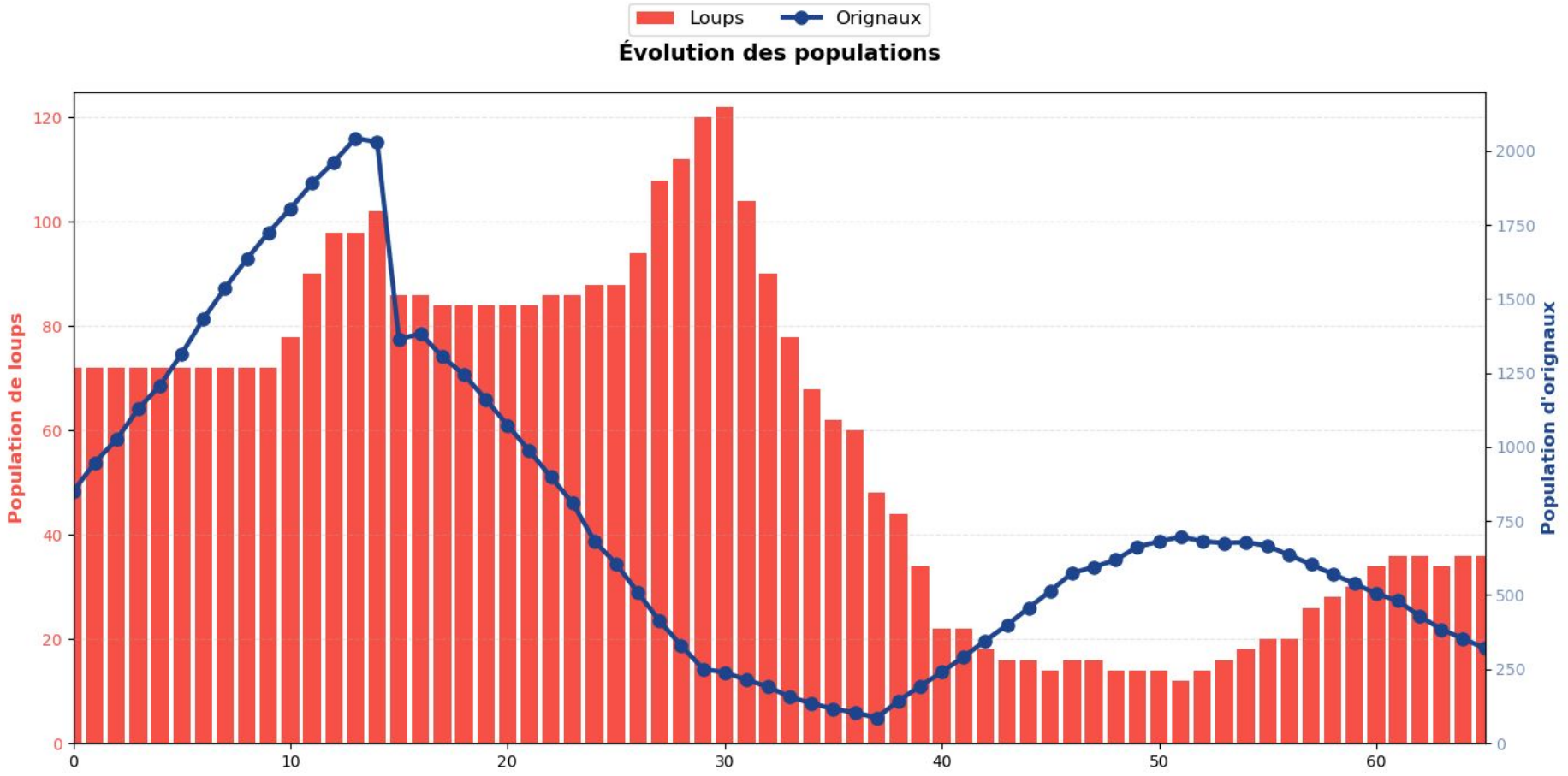

Génération 0 | Orignaux: 947 | Loups: 72
0
10
20
30
40
0
10
20
30
40


Génération 12 | Orignaux: 2042 | Loups: 98
0
10
20
30
40
0
10
20
30
40

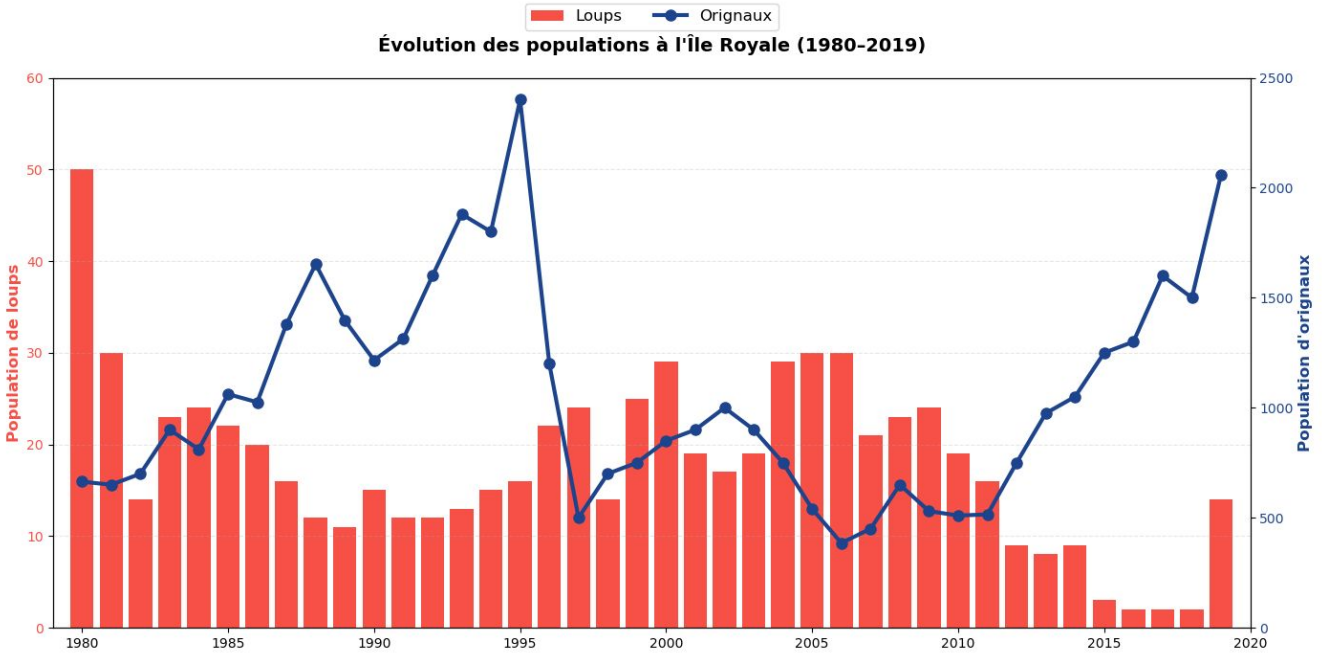

Génération 29 | Orignaux: 239 | Loups: 122
0
10
20
30
40
0
10
20
30
40

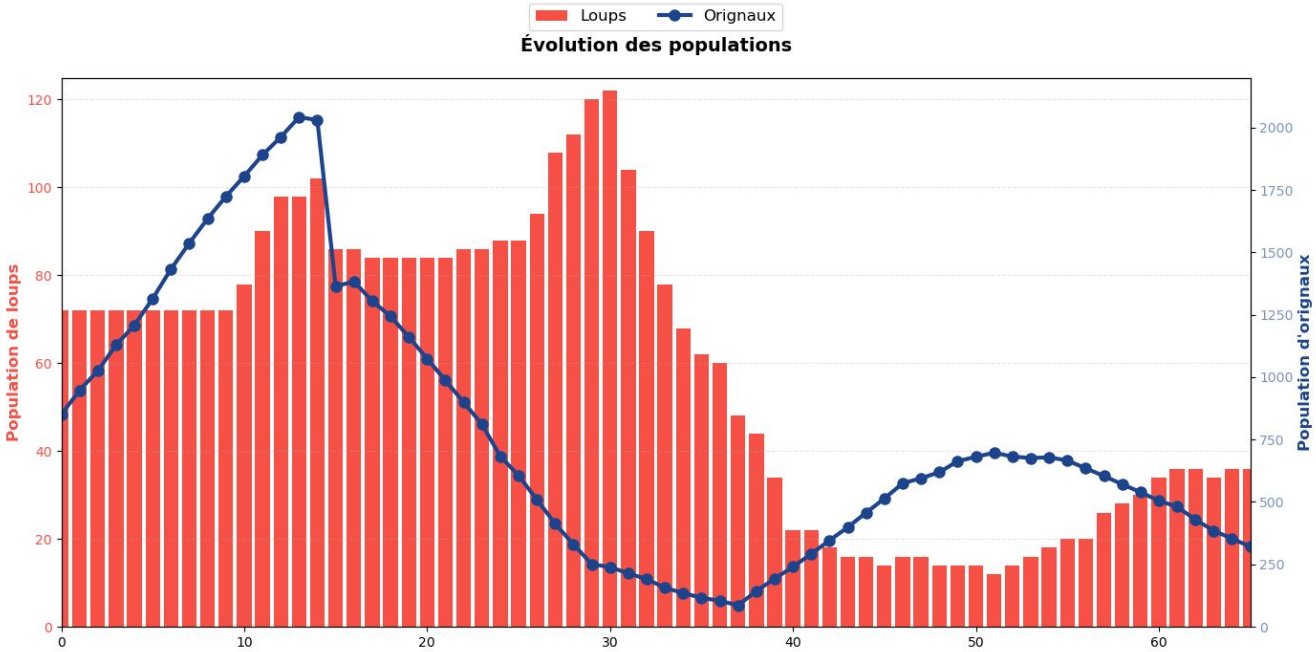

Génération 50 | Orignaux: 697 | Loups: 12
0
10
20
30
40
0
10
20
30
40


Génération 67 | Orignaux: 231 | Loups: 38
0
10
20
30
40
0
10
20
30
40


Loups
Orignaux
Évolution des populations
Population de loups
Population d'orignaux
0
20
40
60
80
100
120
0
250
500
750
1000
1250
1500
1750
2000
0
10
20
30
40
50
60

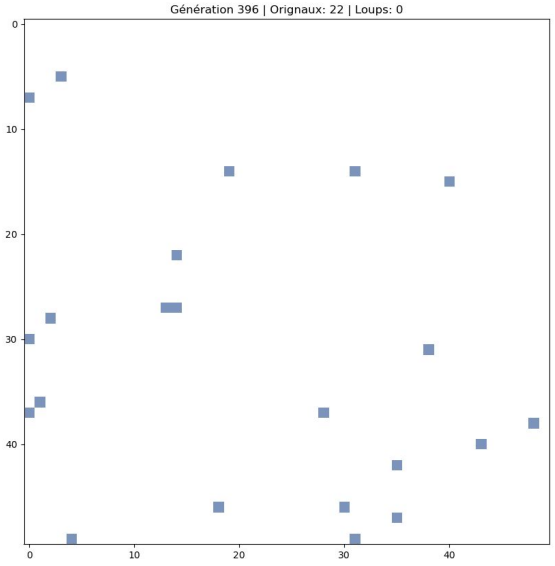
COMPARAISON THÉORIE/PRATIQUE


Loups
Orignaux
Évolution des populations à l'Île Royale (1980–2019)
Population de loups
Population d'orignaux
0
10
20
30
40
50
60
0
500
1000
1500
2000
2500
1980
1985
1990
1995
2000
2005
2010
2015
2020

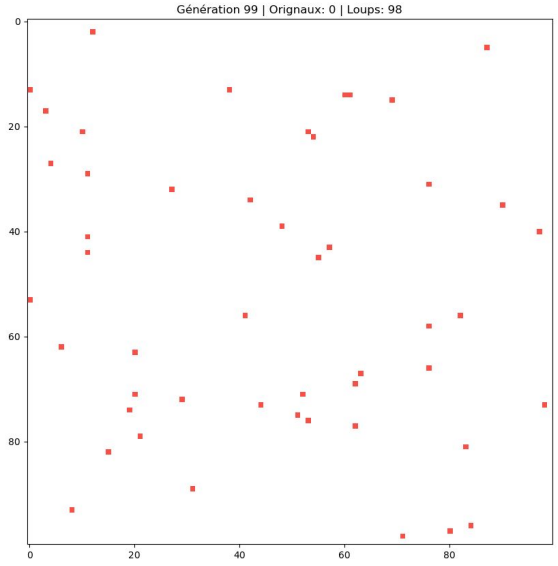

Loups
Orignaux
Évolution des populations
Population de loups
Population d'orignaux
0
20
40
60
80
100
120
0
250
500
750
1000
1250
1500
1750
2000
0
10
20
30
40
50
60

LIMITES DU MODÈLE

- Paramètres fixes.
- Modèle simplifié.



Simulation avec 500 generations



Simulation avec une grille de taille 100

CONCLUSION

- Résultats cohérents avec le réel (Îsle Royale).
- Simulation possible mais de manière simplifiée.

→ Une simulation peut représenter les grandes dynamiques, mais reste une approximation du réel.

MERCI !

Des questions sur mon écosystème ?

RÉFÉRENCES

Figure 1 : https://www.jeulin.net/automates/automate.html
Figure 2 : https://www.michiganpreserves.org/isle-royale-national-park/
Figure 3 : https://www.nps.gov/isro/learn/nature/wolf-moose-populations.htm

ANNEXES

Code utilisé pour la figure 3

```
import matplotlib.pyplot as plt

# Données brutes sous forme de listes
annees = list(range(1980, 2020))
loups = [50, 30, 14, 23, 24, 22, 20, 16, 12, 11, 15, 12, 12, 13, 15, 16,
        22, 24, 14, 25, 29, 19, 17, 19, 29, 30, 30, 21, 23, 24, 19, 16,
        9, 8, 9, 3, 2, 2, 2, 14]
orignaux = [664, 650, 700, 900, 811, 1062, 1025, 1380, 1653, 1397, 1216,
          1313, 1600, 1880, 1800, 2400, 1200, 500, 700, 750, 850, 900,
          1000, 900, 750, 540, 385, 450, 650, 530, 510, 515, 750, 975,
          1050, 1250, 1300, 1600, 1500, 2060]

# Création du graphe
fig, ax1 = plt.subplots(figsize=(14, 7))

# Barres pour les loups
ax1.bar(annees, loups, color="#f65047", width=0.8, label='Loups')
ax1.set_ylabel("Population de loups", color="#f65047", fontsize=12, fontweight='bold')
ax1.tick_params(axis='y', labelcolor="#f65047")
ax1.set_ylim(0, 60)

# Courbe pour les orignaux
ax2 = ax1.twinx()
ax2.plot(annees, orignaux, color="#1d448c", linewidth=3,
        marker='o', markersize=8, markeredgecolor="#1d448c", label='Orignaux')
ax2.set_ylabel("Population d'orignaux", color="#1d448c", fontsize=12, fontweight='bold')
ax2.tick_params(axis='y', labelcolor="#1d448c")
ax2.set_ylim(0, 2500)

# Titre et grille
plt.title("Évolution des populations à l'Île Royale (1980–2019)",
         fontsize=14, pad=20, fontweight='bold')
ax1.grid(axis='y', linestyle='--', alpha=0.3)

# Légende unifiée
lines1, labels1 = ax1.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()
ax1.legend(lines1 + lines2, labels1 + labels2, loc='upper center',
          bbox_to_anchor=(0.5, 1.15), ncol=2, fontsize=12)

# Ajustements finaux
plt.xlim(1979, 2020)
plt.tight_layout()
plt.savefig("stats.png", dpi=100)
plt.show()
```

Code utilisé pour la simulation

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from matplotlib.colors import ListedColormap

# Paramètres
size = 50  # Taille de la grille
n_steps = 100  # Nombre de générations
proie_rep = 0.2  # Taux de reproduction des proies (20%)
seuil_reproduction_loup = 10  # Proies mangées nécessaires pour se reproduire
duree_vie_max1 = 15  # Âge maximal des loups
duree_vie_max2 = 25  # Âge maximal des orignaux
seuil_loups_bas = 50  # Seuil en dessous duquel les orignaux se reproduisent plus
boost_reproduction = 0.8  # Bonus de reproduction quand loups sont rares

# Initialisation
proies = np.random.choice([0, 1], size*size, p=[0.7, 0.3]).reshape(size, size)  # 30% d'orignaux
predateurs = np.where(proies == 1, 0, np.random.choice([0, 2], size=(size, size), p=[0.98, 0.02]))  # 2% de loups
proies_mangees = np.zeros((size, size))  # Compteur de proies mangées par loup
age_predateurs = np.zeros((size, size))  # Âge de chaque loup
age_proies = np.zeros((size, size))  # Âge de chaque orignal
sante_predateurs = np.ones((size, size))  # Santé (1 au début)

# Pour stocker l'évolution des populations
population_proies = []
population_predateurs = []

# Couleurs
cmap = ListedColormap(['white', "#7c94bc", "#f65047"])  # Vide, Proie, Prédateur

def update(frame):
    global proies, predateurs, proies_mangees, age_predateurs, age_proies, sante_predateurs, seuil_reproduction_loup

    nouv_proies = proies.copy()
    nouv_predateurs = predateurs.copy()
    nouv_proies_mangees = proies_mangees.copy()
    nouv_age = age_predateurs.copy()
    nouv_ageo = age_proies.copy()
    nouv_sante = sante_predateurs.copy()
    nouv_seuil_reproduction_loup = seuil_reproduction_loup
```

```
for i in range(size):
    for j in range(size):

        # Comportement des proies
        if proies[i, j] == 1:

            nouv_ageo[i, j] += 1  # Vieillissement

            # Calcul dynamique du taux de reproduction
            taux_actuel = proie_rep
            if np.sum(predateurs) < seuil_loups_bas:
                taux_actuel += boost_reproduction

            # Phase 1 : Reproduction
            if np.random.rand() < taux_actuel and np.sum(proies) < size*size * 0.8:
                ni, nj = np.random.randint(0, size, 2)
                if nouv_proies[ni, nj] == 0:
                    nouv_proies[ni, nj] = 1

            # Phase 2 : Mortalité
            if (nouv_ageo[i, j] > duree_vie_max1):
                nouv_proies[i, j] = 0


        # Comportement des prédateurs
        elif predateurs[i, j] == 2:

            nouv_age[i, j] += 1  # Vieillissement

            # Liste les voisins
            voisins = [(i+di, j+dj) for di in [-1, 0, 1] for dj in [-1, 0, 1]
                    if 0 <= i+di < size and 0 <= j+dj < size and (di != 0 or dj != 0)]
            np.random.shuffle(voisins)
            deplacement = False

            # Phase 1 : Chasse
            for ni, nj in voisins:
                if nouv_proies[ni, nj] == 1:
                    nouv_proies[ni, nj] = 0  # Mange la proie
                    nouv_proies_mangees[i, j] += 1
                    nouv_sante[i, j] = nouv_sante[i, j] + 0.1  # Améliore la santé
                    deplacement = True
                    break
```

```
            # Phase 2 : Déplacement
            if not deplacement:
                nouv_sante[i, j]= nouv_sante[i, j] -0.25  # Pénalité santé si ne mange pas
                for ni, nj in voisins:
                    if nouv_predateurs[ni, nj] == 0:
                        nouv_predateurs[i, j] = 0
                        nouv_predateurs[ni, nj] = 2
                        nouv_proies_mangees[ni, nj] = nouv_proies_mangees[i, j]
                        nouv_age[ni, nj] = nouv_age[i, j]
                        nouv_sante[ni, nj] = nouv_sante[i, j]
                        break
            if np.sum(predateurs) < 25:  # Seuil critique de loups
                nouv_sante[i, j] = max(1.2, nouv_sante[i, j] + 0.3)  # Bonus santé
                nouv_seuil_reproduction_loup = 4  # Réduction du seuil nécessaire

            # Phase 3 : Reproduction
            if (nouv_proies_mangees[i, j] >= nouv_seuil_reproduction_loup and
                nouv_sante[i, j] > 1.2 and
                np.random.rand() < 0.4):
                ni, nj = np.random.randint(0, size, 2)
                if nouv_predateurs[ni, nj] == 0:
                    nouv_predateurs[ni, nj] = 2
                    nouv_proies_mangees[ni, nj] = 0  # Proies mangées par le nouveau loup
                    nouv_age[ni, nj] = 0
                    nouv_sante[ni, nj] = 0.8  # Santé initiale réduite (consanguinité)
                    nouv_proies_mangees[i, j] = 0  # Reset parent

            # Phase 4 : Mortalité
            if (nouv_age[i, j] > duree_vie_max1 or
                nouv_sante[i, j] <= 0.4):
                nouv_predateurs[i, j] = 0

    # Mise à jour globale
    proies, predateurs = nouv_proies, nouv_predateurs
    proies_mangees, age_predateurs, sante_predateurs, age_proies, seuil_reproduction_loup= nouv_proies_mangees, nouv_age,
nouv_sante, nouv_ageo, nouv_seuil_reproduction_loup
```

```
    # Visualisation
    img.set_array(proies + predateurs)
    ax.set_title(f"Génération {frame} | Orignaux: {np.sum(proies)} | Loups: {np.sum(predateurs)}")
    # Stockage des données de population
    population_proies.append(np.sum(proies))
    population_predateurs.append(np.sum(predateurs))

    return img

# Animation
fig, ax = plt.subplots(figsize=(10, 10))
img = ax.imshow(proies + predateurs, cmap=cmap, interpolation='nearest')
ani = animation.FuncAnimation(fig, update, frames=n_steps, interval=100)

ani.save('parfait4.gif', writer='pillow', fps=10, dpi=100)

plt.show()


# Création du graphe
fig2, ax1 = plt.subplots(figsize=(14, 7))

# Barres pour les loups
ax1.bar(range(len(population_predateurs)), population_predateurs, color="#f65047", width=0.8, label='Loups')
ax1.set_ylabel("Population de loups", color="#f65047", fontsize=12, fontweight='bold')
ax1.tick_params(axis='y', labelcolor="#f65047")
ax1.set_ylim(0, 125)

# Courbe pour les orignaux
ax2 = ax1.twinx()
ax2.plot(range(len(population_proies)), population_proies, color="#1d448c", linewidth=3,
        marker='o', markersize=8, markeredgecolor="#1d448c", label='Orignaux')
ax2.set_ylabel("Population d'orignaux", color="#1d448c", fontsize=12, fontweight='bold')
ax2.tick_params(axis='y', labelcolor="#7c94bc")
ax2.set_ylim(0, 2200)
```

```
# Titre et grille
plt.title("Évolution des populations",
          fontsize=14, pad=20, fontweight='bold')
ax1.grid(axis='y', linestyle='--', alpha=0.3)

# Légende unifiée
lines1, labels1 = ax1.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()
ax1.legend(lines1 + lines2, labels1 + labels2, loc='upper center',
           bbox_to_anchor=(0.5, 1.15), ncol=2, fontsize=12)

# Ajustements finaux
plt.xlim(0, 65)
plt.tight_layout()
plt.savefig("parfaitx3.png", dpi=100)
plt.show()
```