

# SCI : Simulation Centrée Individu

## Axel Thavisouk

### Lancer les différentes parties

Le TP a été réalisé en langage python et l'interface graphique Tkinter il faut donc Tkinter (qui peut parfois ne pas être installé)

Pour lancer les différentes parties il faut exécuter le main correspondant:

Pour particules, lancez: - python3 mainParticules.py

Pour Wator, lancez: - python3 mainWator.py

Pour Pacman, lancez: - python3 mainPacman.py

### Génération des expérience

Pour créer les 2 graphes du TP Wator, il faut activer la trace et le piper dans le fichier poisson.data situé dans le dossier Expe. Il ne reste plus qu'à exécuter la commande suivante:

- gnuplot result.plt Python n'est pas le meilleur langage pour faire ça. Si l'on cherche à aligner les 2 courbes on peut ajuster la ligne:
- set y2range [0:3200]

ici c'est le 3200

Pour créer les graphes pour particules, il faut piper le résultat de la trace de particules dans des fichier. dans le fichier particules.plt il faut modifier la dernière lignes pour correspondre à vos fichier créé.

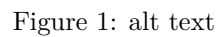
Après le plot de la dernière ligne on peut rajouter de multiples lignes

- '50x50.data' using 1:2 with lines title '50x50' sur lesquels il faut modifier ce qui est entre quote. (le nom du fichier et le titre)

### Eléments de configurations en communs

- torus : 1 si torique, 0 sinon
- gridSizeX : nombre de cases en abscise
- gridSizeY : nombre de cases en ordonnée
- boxSize : la taille des cases
- scheduling : les ordonnancements (equitable, sequentiel, random)
- nbTicks : le nombre de tours (-1 pour infini)

- # Diagramme UML



## Partie particules

la partie particules est dans le package Particules. Il contient les fichiers suivants:

- Particules.py: Contient la classe de l'agent Particule
- sma.py: Contient le core de la partie particule
- parameter.json: le fichier de config de la partie particule

## Config

- nbParticules : le nombre de particules

## Questions

### Questions 1

On peut faire tourner, sur un ordinateur ayant un processeur Intel® Core™ i5-7200U CPU @ 2.50GHz  $\times$  4 et 8 Go de RAM, 120000 particules qui bougent à une fréquence d'environ 1,5 tour/seconde

### Question 2

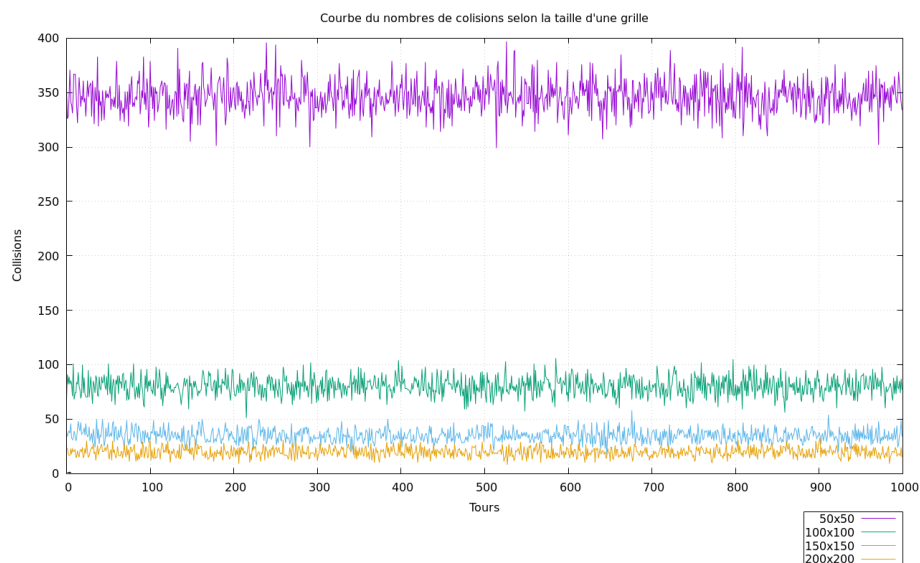


Figure 2: alt text

Cette expérience a été réalisée pour 1000 Particules sur différentes tailles de grilles. On peut observer que la différence du nombre de collisions entre la grille

de taille 50x50 et 100x100 est énorme. si l'on compare la différence entre la grille de 150x150 à celle de 200x200 on voit qu'il y a une différence mais pas aussi énorme. On y a pourtant ajouté plus de cases pour la 2ème comparaison que la 1ère.(7500 contre 17500)

### Question 3

Dans le cas où l'on ne fait rien lorsqu'une particule est présente, on supprime les particules où l'on veut aller. si l'on utilise une deuxième grille tampon pour stocker les particules qui se font supprimer on peut contourner le problème et dans ce cas il n'y a plus aucune collision.

Dans le cas où j'inverse la direction lors d'une collision, j'ai eu d'abord des problèmes où certaines particules supprimaient d'autres qui étaient dans l'environnement, mais en utilisant la même solution qu'en ne faisant rien on obtiens des mouvements assez étranges car les particules que l'on rencontre ne sont donc pas du tout affectées par les collisions qui n'ont pas été réalisées par eux.

Le cas où l'on échange les pas des deux particules qui collisionnent est le comportement qui donne le meilleur rendu car en effet les deux particules sont dans ce cas affecté par la collision et il n'y a plus besoin de tableau tampons car il n'est plus possible d'avoir deux particules différentes sur la même case.

## Partie Wator

### Package

la partie Wator est dans le package Wator. Il contient les fichiers suivants:

- fish.py: Contient la classe de l'agent Fish
- shark.py: Contient la classe de l'agent Shark
- wator.py: Contient le core de la partie wator
- wator.json: le fichier de config de la partie wator
- watorEnv.py: l'environnement de Wator

### Config

- waterColor : la couleur du background
- fishColor : la couleur du poisson
- sharkColor : la couleur des requins
- newFishColor : la couleur des nouveaux poissons
- newSharkColor : la couleur des nouveaux requins
- fishBreedTime : temps de fécondation pour les poissons
- sharkBreedTime : temps de fécondation pour les requins

- sharkStarveTime : temps avant de mourir de faim pour les requins,
- nbShark : le nombre de requins
- nbFish : le nombre de poissons

## Questions

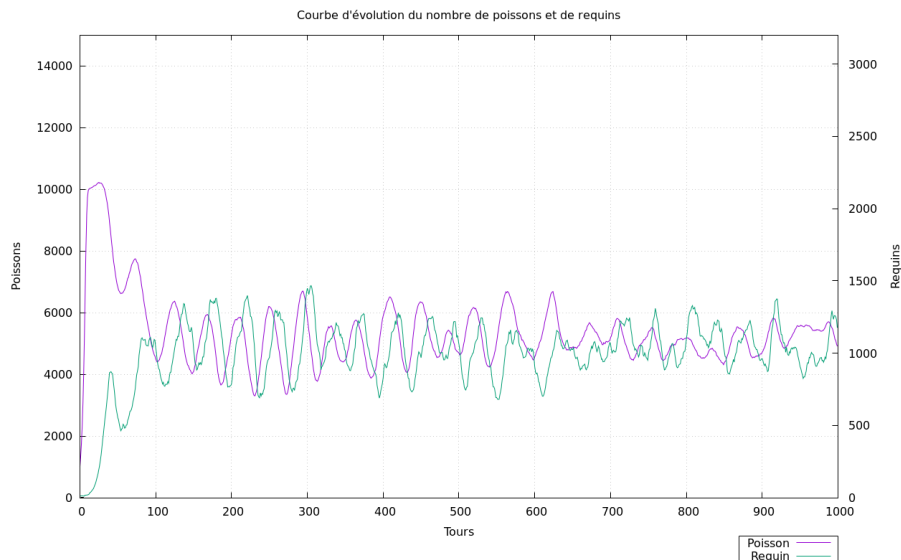
### Question 1

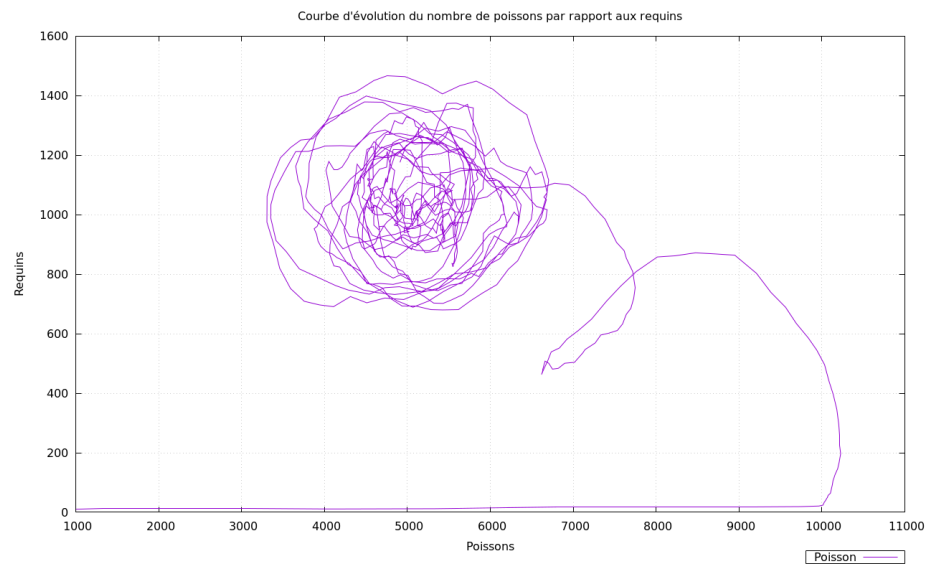
Il est préférable d'initialiser les différents compteurs de manière aléatoire car lorsque l'on fixe ces compteurs aux même valeurs, on obtient des poissons et leurs enfants qui naissent et qui meurent en même temps. On peut le voir par la croissance en bloc sur la grille lorsque l'on le lance ainsi.

### Question 2

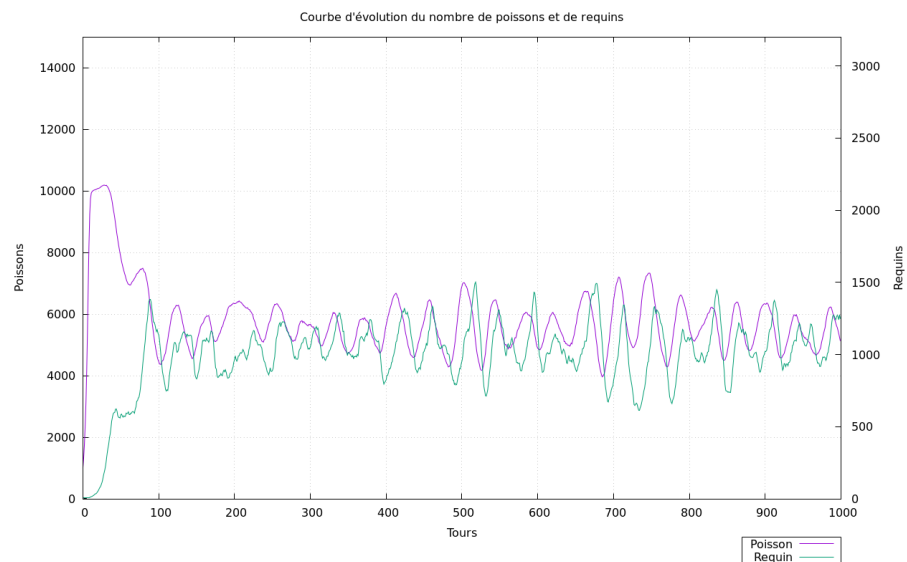
Lorsque l'on fixe une action par tick on bloque lorsque la grille est remplie de poissons car ils ne peuvent plus se reproduire car les poissons se seront reproduits là où les requins étaient avant

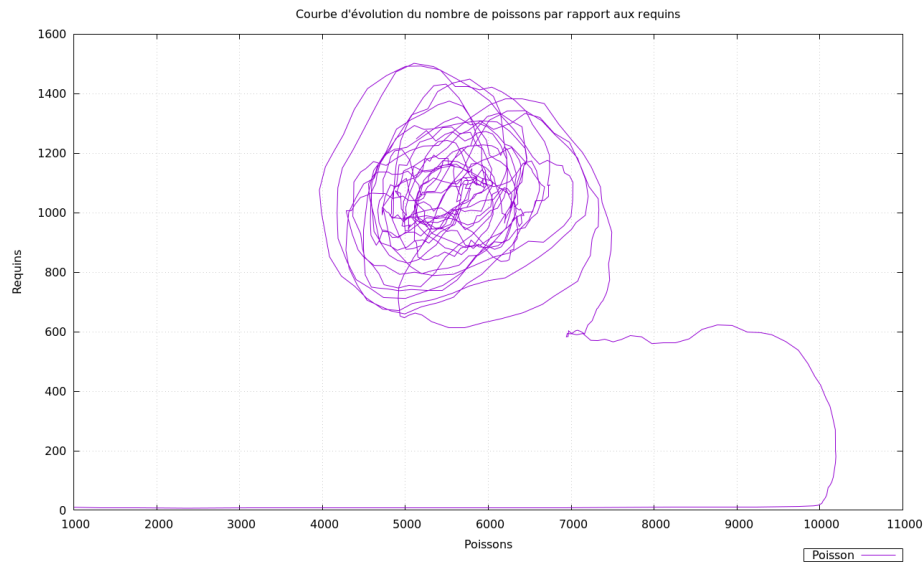
Se reproduire en bougeant donne le meilleur résultat, ils donnent aux requins une meilleure survie et les poissons se reproduisent assez rapidement aussi, on voit sur la courbe d'évolution que les poissons ont l'air de se stabiliser un peu mieux sur la fin





Se reproduire en mangeant ne permet pas aux poissons de stabiliser leurs populations comme le faisait le comportement précédent





## Partie Pacman

### Package

la partie Pacman est dans le package Pacman. Il contient les fichiers suivants:

- avatar.py: Contient la classe de l'agent avatar
- defender.py: Contient la classe de l'agent Defender
- hunter.py: Contient la classe de l'agent Hunter
- wall.py: Contient la classe de l'agent Wall
- winner.py: Contient la classe de l'agent Winner
- pacman.py: Contient le core de la partie pacman
- maze.json: le fichier de config de la partie pacman

### Config

- speedHunter : nombre de tours nécessaires pour le mouvement d'un hunter
- speedAvatar : nombre de tours nécessaires pour le mouvement de l'avatar
- nbHunter: nombre de hunter
- nbWall : pourcentage de mur, ignoré si maze à 1
- defenderLife : nombre de tours avant la disparition du defender,
- invincibility : nombre de tour de l'invincibilité,
- maze : 1 pour obtenir un labyrinthe, 0 sinon
- erosionLabyrinthe : un nombre entre 0 et 1 afin d'éroder le labyrinthe pour ne pas avoir qu'un seul chemin

### **Les touches**

- Les touches A et Z permettent d'accélérer et décélérer les Hunters
- Les touches O et P permettent d'accélérer et décélérer l'Avatar
- Les touches W et X permettent d'accélérer et décélérer le jeu
- La touche Espace permet de mettre en pause le jeu et de le reprendre

### **les limites et les difficultés rencontrée**

J'ai été énormément limité par le langage python, pour ce genre d'application, il a énormément de mal sur la gestion de la mémoire, les vitesse des accès. J'ai eu énormément de problème lors de la conception objet. Première fois que j'en fait dans ce langage.