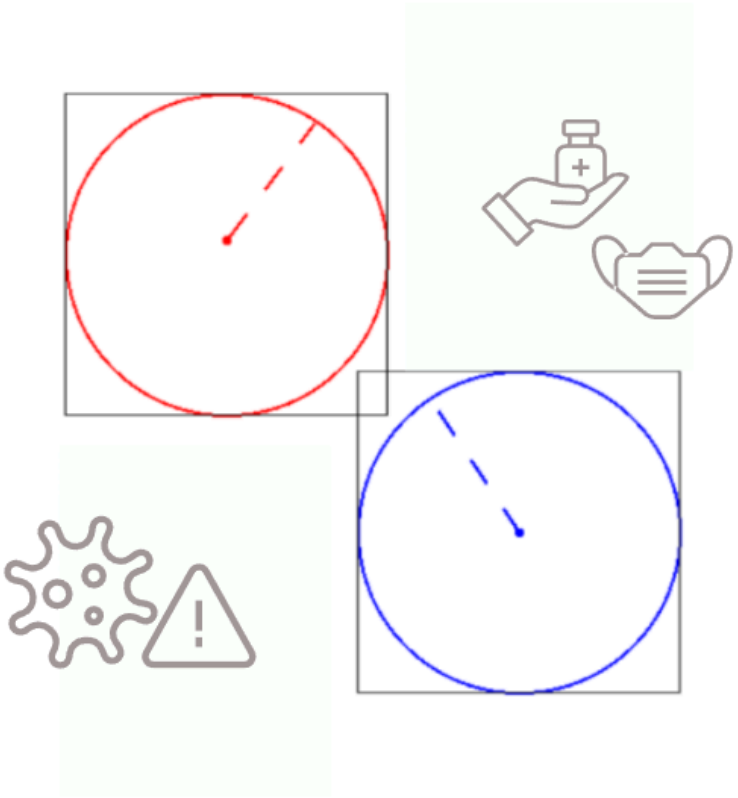
LIGNIER Noémie
TPB

Optimisation d'algorithmes pour une simulation de diffusion du Covid



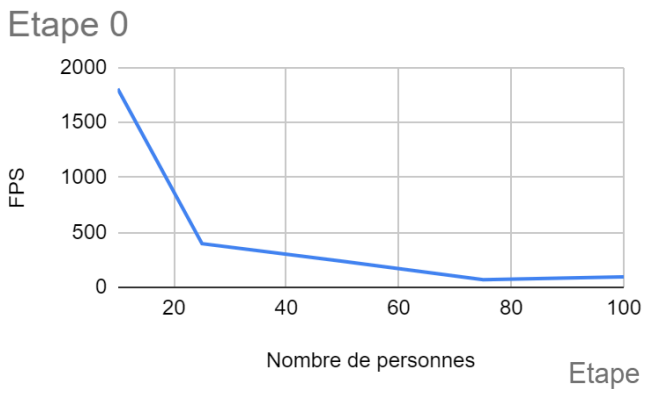
SAÉ 1.02 -Comparaison d'approches algorithmiques

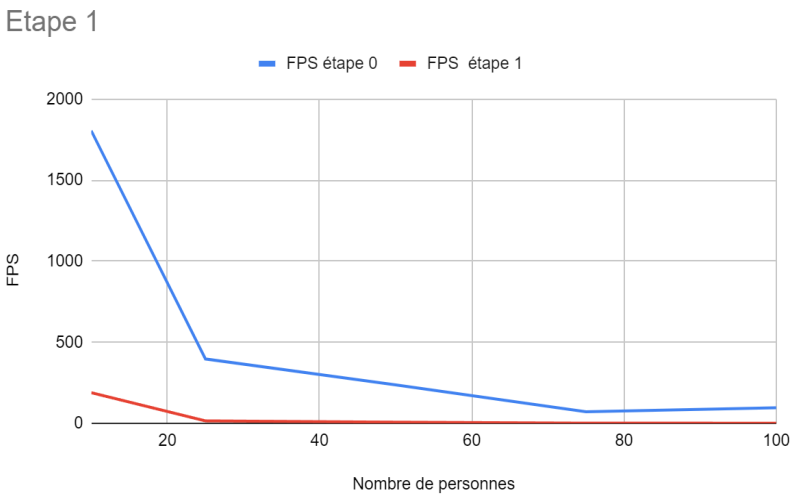
Institut Universitaire de Technologie LITTORAL CÔTE D'OPALE

LIGNIER Noémie
TPB

Étape 1 : Optimisation de l'affichage

Pour commencer, afin de fluidifier la simulation j'ai accompli une première étape visant à rectifier l'affichage de la simulation.
Pour réaliser cette première étape qui va concerner l'optimisation de l'affichage j'ai fait en sorte que la mise à jour de l'affichage se réalise une fois que tout le monde est déplacé et pas lorsque chaque personne se déplace c'est-à-dire que j'ai changé la mise à jour des fps (frame per second) quand toutes les personnes ont bougé. Pour ce faire il m'a suffit d'accéder à la partie principale de la simulation dans le code c'est-à-dire à la "main loop" pour y changer le nombre de frames (images) ajoutés à chaque occurrences de la boucle. Il m'a donc fallu sortir de la boucle l'ajout de frames qui est à la ligne 93 du programme 'pandemic' et à la ligne 50 du programme 'bench' en changeant l'indentation.Suite à ça les mise à jour de l'affichage ne se font que lorsque la boucle (qui s'occupe des déplacements) est terminée.
Après ce changement et l'exécution de cette étape, on peut constater que la simulation n'est pas forcément plus fluide mais permet de réduire le nombre de fps a chaque tour.



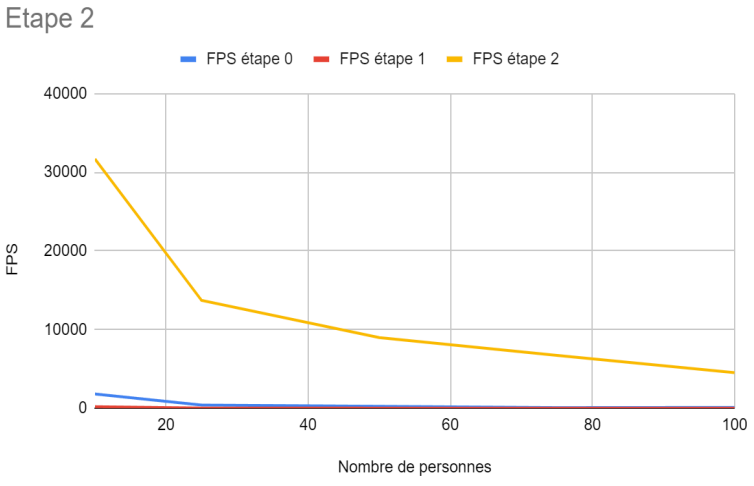


Étape 2 : Optimisation mathématique

Cette seconde étape vise à optimiser la simulation d'un point de vue mathématique.

En effet, la simulation passe le principal de son temps à tester des intersections entre des cercles. Il était donc important d'optimiser l'algorithme dans la fonction *circleCollision* du fichier *engine.py* qui vérifie si deux cercles se touchent ou pas.
Cette fonction utilise un grand nombre de calculs pour arriver à son résultat et utilise des fonctions mathématiques assez lourdes comme l'utilisation de la racine carré pour calculer la distance entre deux points.
L'objectif était donc de réduire ce nombre de calculs inutiles et de trouver une solution afin de ne pas utiliser la fonction math.sqrt inscrite dans le code.De plus la fonction parcours la surface des cercles pour choisir un point du cercle selon un angle qui change à chaque itération de la boucle afin de vérifier s'il appartient à un deuxième cercle mais cela est inutile puisqu'il existe une solution plus simple utilisant moins de calculs.
En effet il faut prendre le centre du premier cercle, prendre celui du second et ensuite calculer la distance entre les deux puis on vérifie si la distance est plus petite que le rayon et si c'est le cas on dit que la collision a bien lieu.
Donc dans un premier temps j'ai retiré la boucle qui servait à parcourir la surface du cercle selon un angle pour ne calculer qu'une seule fois les coordonnées et j'ai changé ce calcul pour alléger la façon d'obtenir les coordonnées.
J'ai donc aussi décidé de supprimer la ligne de code qui utilisait la fonction racine carré et j'ai changé le calcul en y laissant la distance au carré pour ne pas avoir à utiliser la racine carré.
Ensuite j'ai changé le test dans la condition pour comparer la distance et le rayon (tous les deux au carré).

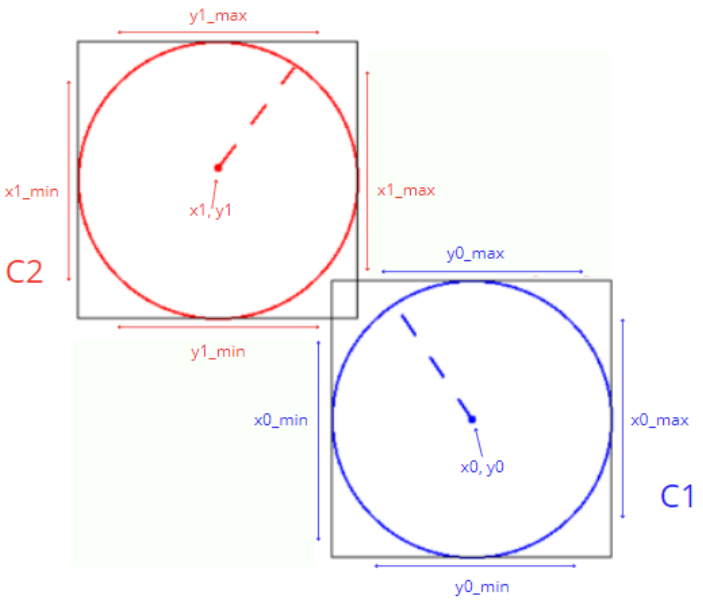
Une fois ces changements réalisés on remarque que la fonction est bien moins lourde et est beaucoup plus optimisée du fait d'avoir retiré tous ces calculs inutiles.Lorsque que l'on exécute le fichier *bench.py* on peut constater que le nombre de frames per second est bien plus adapté à la simulation et celle ci est plus fluide lors de son lancement.

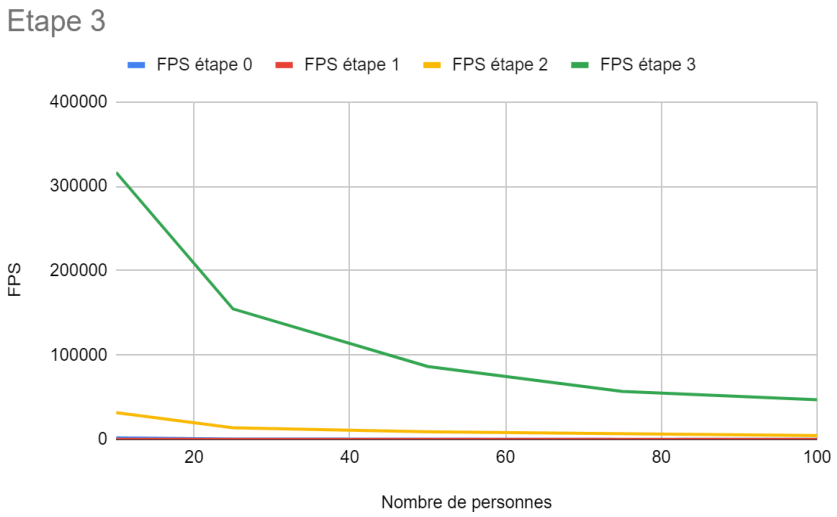


Étape 3 : Optimisation par approximation

Après avoir réussi à optimiser l'algorithme de test d'intersection des cercles à l'étape précédente, on a pu voir une forte hausse de la performance de la simulation.Cependant, l'algorithme se base encore sur un calcul de distance et ce calcul nécessite des opérations mathématiques très lourdes.

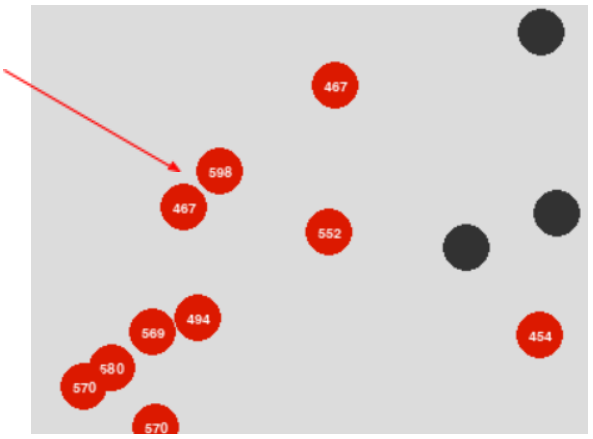
Dans cette étape, le but est d'optimiser cet algorithme en supprimant les calculs complexes de la fonction de test d'intersection.Pour parvenir à améliorer les performances j'ai donc utilisé un système de boîtes englobantes qui vise à prendre le plus petit rectangle qui encadre les cercles afin de réduire le nombre de calculs pour les tests d'intersections.Ce système est plus performant que le précédent.Pour pouvoir l'utiliser, je me suis donc servi des coordonnées des centres de chaque cercles et j'ai calculé la hauteur et la longueur maximale de chaque boîte(rectangle) autour d'un cercle.Pour ne pas avoir à calculer les boîtes englobantes à chaque test d'intersection il faut calculer chaque box pour le nombre total de personnes dans la simulation et je mets ces coordonnées dans une liste à l'aide d'une fonction *list_bording_box* dans laquelle je remplir une liste nommée box avec des tuples contenants les coordonées d'un cercle. Dans la fonction *circleCollision* je n'ai donc plus à calculer ceci et je suis venue faire une comparaison entre les boîtes des cercles, par exemple je vais venir regarder si le haut de la boite du cercle c1 (y0_max) est à une plus haute coordonnée que le bas de la boite du cercle c2 (y1_min) (voir schéma ci-dessous). Si c'est le cas, cela veut donc dire que les deux boîtes se touchent et on dit que la collision à lieu.On fait donc le test avec les différents côtés des deux boîtes pour vérifier s'il y a bien une intersection entre eux et on retourne vrai si c'est le cas.





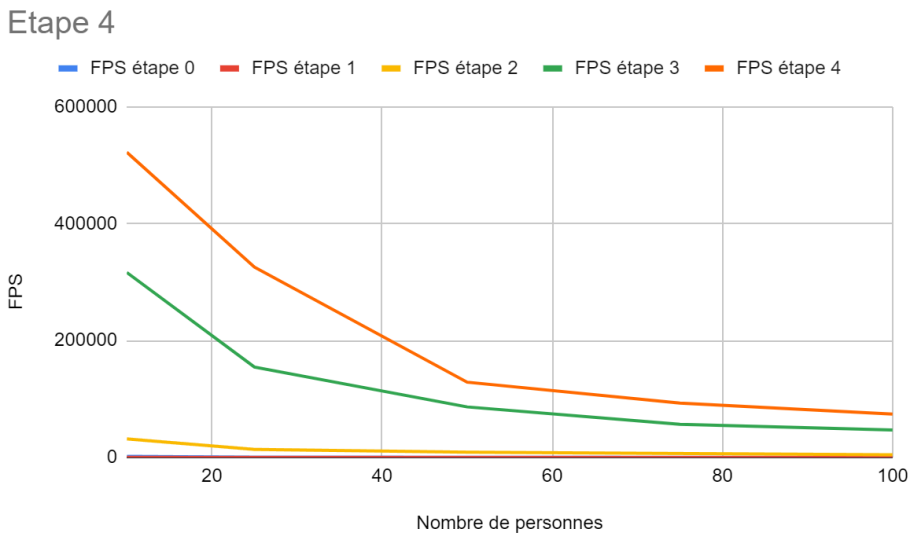
Quel est l'inconvénient de cette méthode ? Argumenter et illustrer vos propos.

L'inconvénient de cette méthode est que les collisions ont lieux même lorsque les deux cercles ne touchent pas réellement.En effet on peut voir en lançant la simulation que lorsque la personne infectée se rapproche de d'une personne en bonne santé celle-ci devient infectée malgré le fait qu'elle ne croise pas la personne malade.
On peut voir sur l'image que les deux personnes sont proches et deviennent infectées sans se toucher.C'est un inconvénient puisque ça ne représente pas vraiment la réalité et la simulation n'est pas



Étape 4 : Optimisation logique/algorithmique

En utilisant les boîtes englobantes à l'étape précédente, on a remarqué une nouvelle forte hausse de la performance de la simulation.
Dans l'état actuel de la simulation, lorsqu'une personne est déplacée, un calcul d'intersection est fait avec toutes les autres personnes. Or, seules les personnes infectées peuvent contaminer les personnes en bonne santé.
Pour optimiser ce code il a donc fallu changer une condition dans la fonction *computeCollissions* qui faisait en sorte que les test ne soit fait qu'avec toutes les autres personnes hormis eux même.En effet pour que le calcul d'intersection ne soit fait qu'avec les personnes infectées j'ai donc ajouté une deuxième condition dans le "si" de la fonction, cette condition consiste à vérifier si la deuxième personne est une personne infectée ou non.Si la deuxième est infectée alors on applique le calcul d'intersection et sinon non.Cette méthode va limiter le nombre de calcul d'intersection à seulement ceux qui sont utiles et va donc nous faire gagner d'avantages de fps.
Après l'exécution de la simulation, on peut à nouveau constater une augmentation des performances.Le graphique ci-dessous représente bien l'évolution des performances depuis l'étape d'origine et on peut remarquer que le nombre de frames per second (fps) est beaucoup plus important.



Conclusion

In order to get results on the number of healthy people, infected or cured as quickly as possible, I had to perform different steps that consisted in obtaining optimizations by modifying what already existed in the simulation. All these independent steps allowed me to differentiate algorithmic strategies to solve the same problem.

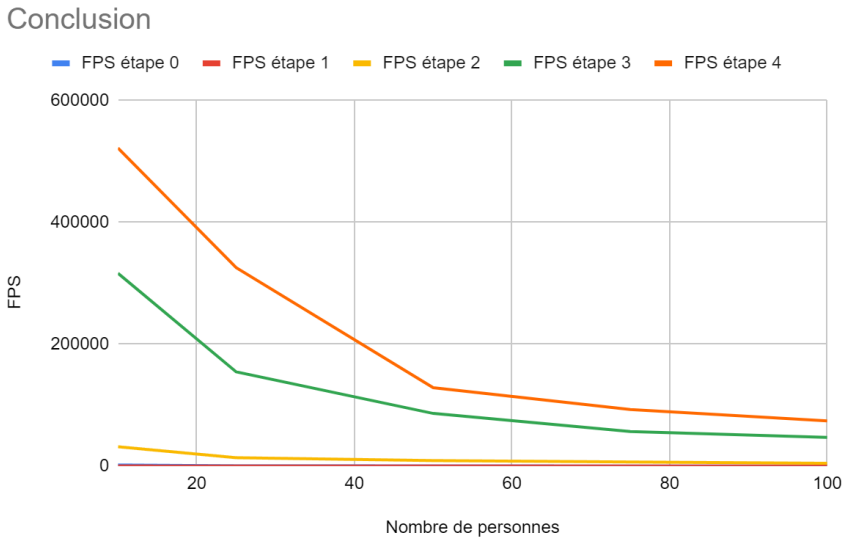
First of all I made a first optimization, during the first step, which consisted in streamlining the simulation by correcting the display of it. That's when we noticed that it was not a very useful optimization since this is even slower than the basic step but thanks to this optimization I was able to solve the problem related to the display so that the update is done only when everyone is moved and not when each person moves.

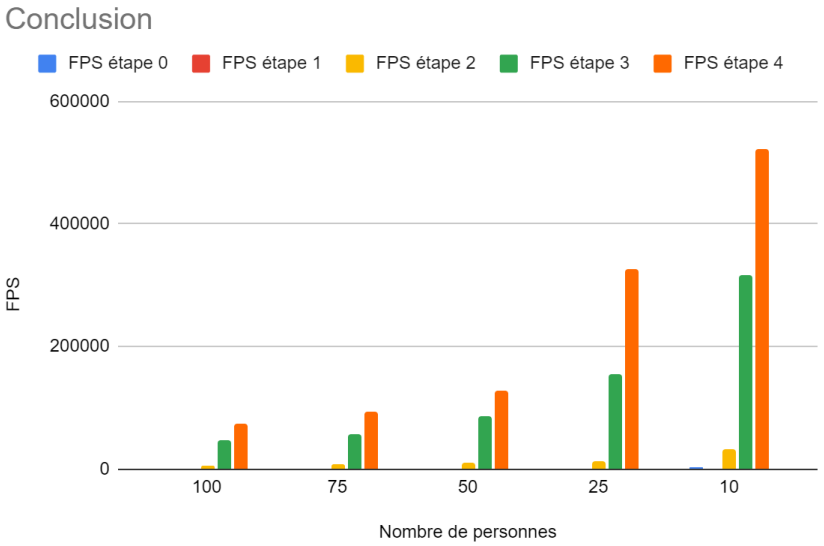
Then I became interested in the second step which aimed to optimize the simulation from a mathematical point of view.As soon as I was able to find the algorithmic strategy and as I ran the program again we could see a big change in the number of fps, indeed the reduction of heavy calculations such as the math.sqrt function for collisions allowed to simplify the code and thus greatly improve the operation and speed of the simulation. The passage through this stage allowed me to gain a lot in fps.

After I had to look at another problem in step 3. Indeed the algorithm was still based on a distance calculation and this calculation required many mathematical operations.By adding bounding boxes we were able to limit the use of this calculation and the algorithm was much simpler and clearer to use. As in the other stages, by launching the simulation we can see a clear improvement in the number of fps by arriving at 47,000 fps per 100 people.The simulation is much faster and we get the results in just a few seconds compared to the first step where we had to wait several minutes

Eventually we could still get performance increases by only doing crash tests for infected people. The purpose of step 4 was precisely to address this problem. So I changed a condition in the function that dealt with it. Following this modification the number of calculations is minimized and the performance is increasing and we can see on the graph that for a simulation of 100 people we reach almost 80,000 FPS.

Each step allowed us to obtain performance increases and to gain fluidity which proves that the code of the algorithms have been optimized and improved to have the results as quickly as possible.We therefore went from having to wait several minutes to the base step a few seconds for the very last step.





The graph above represents the number of fps for each step according to different numbers of people. We can see that step 0 and step 1 do not appear because the number of fps is too small compared to the other steps.To realize, I added a graph with a smaller scale to better see the evolution of steps 1 and 2.