SAE R1.02- Comparaison d’approches algorithmiques

Compte rendu de SAE

Étape 1 : Optimisation de l'affichage

Problème initial :

Le problème de cette première étape, était que l'affichage du déplacement de chaque personne est actualisé à chaque déplacement. Or, il serait plus judicieux de faire cette actualisation en même temps pour chaque cellule.

Afin de résoudre ce souci, il a fallu dans un premier temps modifier le programme principal :

Pandemic.py (lines : 74 – 96)

```
while not endSimulation:

        for j in range (0, len(persons)):
            # update of the person : direction and
health sttae
            engine.update(persons[j])

            # find collisions between persons
            colls = engine.computeCollisions(persons,j)

            # contaminated persons infect healthy
persons
            engine.processCollisions(persons, colls)

            # draw scene
            draw(scene,font,persons)
            # display update
            deltaTime = clock.tick()
            pygame.display.update()
            frameNumber += 1
            # time.sleep(0.2)

        # end if there isn't infected person
        endSimulation = engine.endSimulation(persons)
```

Afin que le programme réalise tous les calculs de chaque cellule avant de TOUS les déplacer, il fallait simplement dés indenter les lignes 86 à 93 (marqués par une ligne rouge au-dessus).

De ce fait, la boucle « for » s'exécute et calcule les déplacements puis s'arrête à la ligne 84.

Certes, après modifications, les programmes ont beaucoup moins de FPS (de 208FPS à 2FPS). Néanmoins, le programme est plus logique et plus fluide.

Programme performances :

Bench.py (lignes 37 à 53)

```
# main loop
    while not endSimulation:

        for j in range (0, len(persons)):
            # update of the person: direction and
health state
            engine.update(persons[j])

            # find collisions between persons
            colls = engine.computeCollisions(persons,j)

            # contaminated persons infect healthy
persons
            engine.processCollisions(persons, colls)

        frameNumber += 1


        endSimulation = engine.endSimulation(persons)
```

Dans le benchmark Il suffit également de dés cémenter une ligne (la ligne 50) marquée par le trait rouge.

Résultats :

	FPS (pour 100 individus)	
	Pandemic.py	Bench.py
Sans modifications	208.4678952601389	269.5057459984188
Avec modifications	2.625604113222217	2.5769089990216796

Étape 2 : Optimisation mathématique

Problème rencontré :

Grâce ou plutôt à cause d'une personne dont je terrerais le nom, le code de la fonction circleColision du fichier engine.py était remplie de formules mathématiques incompréhensibles (pour moi) et qui mangeaient petit à petit les composants de mon ordinateur.

C'est donc pourquoi j'ai décidé de tout supprimer et de tout recommencer !

Solution programme principal :

J'ai donc réécrit entièrement la fonction (en gardant tout de même les paramètres d'entrée).

Engine.py

```
def circleCollision(c1, c2):
    '''
    Determines if two circles intersect or not
    '''
    # Retrieve the coordinates of the two circles:
    # Circle 1
    PositionXCircle1 = c1[0]
    PositionYCircle1 = c1[1]
    # Circle 2
    PositionXCircle2 = c2[0]
    PositionYCircle2 = c2[1]

    # This condition checks whether:
    # The absolute position of the first and second circles is less than twice the radius of the circle, to check if the circles are touching.
    if abs(PositionXCircle1 - PositionXCircle2) < constants.PERSON_RADIUS_2 and abs(PositionYCircle1 - PositionYCircle2) < constants.PERSON_RADIUS_2:
        return True
    else:
        return False
```

J'ai donc dans un premier temps récupéré les coordonnées x,y des deux cercles que j'ai stockés dans des variables (voir traits verts).

Ensuite, j'ai déclaré une condition qui vérifie si la position sous forme de valeur absolue des deux cercles est inférieure au double du rayon du cercle (voir trait bleu).

Si la valeur est vraie, cela retourne Vrai et inversement.

Au revoir les cosinus et racines carrées !

Benckmark :

Il n'y a rien à modifier dans ce programme car le benchmark appelle la fonction circleCollision que l'on vient de modifier.

	FPS (pour 100 individus)	
	Pandemic.py	Bench.py
Sans modifications	2.625604113222217	2.5769089990216796
Avec modifications	242.17796248420123	394.5415081026605

Étape 4 : Optimisation logique/algorithmique

Problème rencontré :

Jusqu'à lors, le programme vérifiait un tas de choses sur chacun des 100 individus. Afin de résoudre ce problème, j'ai divisé le nombre par 2 progressivement de personnes à calculer.

Solution :

J'ai donc ajouté une condition dans le fichier engine.py. Cette condition vérifie si une personne est contaminée ou non.

Engine.py (ligne 73 – 85)

```
# main loop
    while not endSimulation:

        for j in range (0, len(persons)):
            # update of the person : direction and
health sttae
            engine.update(persons[j])

            # find collisions between persons
            if persons[j][2] == constants.INFECTED:
                colls =
engine.computeCollisions(persons,j)

            # contaminated persons infect healthy
persons
                engine.processCollisions(persons,
colls)
```

La modification du benchmark est exactement la même que celle du fichier engine.py.

Performances :

	FPS (pour 100 individus)	
	Pandemic.py	Bench.py
Sans modifications	242.17796248420123	394.5415081026605
Avec modifications	444.5600700509596	1135.7047806533549

Exercises	BEFORE MODIFICATIONS		AFTER MODIFICATIONS	
	PANDEMIC	BENCH	PANDEMIC	BENCH
Start	208.4678952601389	269.5057459984188	2.625604113222217	2.5769089990216796
1	2.625604113222217	2.5769089990216796	242.17796248420123	394.5415081026605
2	242.17796248420123	394.5415081026605		
3			242.17796248420123	394.5415081026605
4	242.17796248420123	394.5415081026605	444.5600700509596	1135.7047806533549

We can see that over the course of the exercises, the performance has greatly increased. We can see that for the benchmark, it went from 200 FPS to 1135 FPS! While for the graphics program pandemic.py, it went from about 200 FPS to 444 FPS.

The first algorithm allowed me to make all the individuals move at the same time just by moving a line (to take it out of the loop). After this modification, I could contrast a slowdown of the program. However, it was synchronised.

Then, in the second program, I changed a function that was considered to be "null". After this modification, which was much more efficient, and much faster, it went from 2 to 242 FPS, which allowed me to find my starting point in terms of frames per second.

Finally, the last exercise allowed me to eliminate and sort out the people to be calculated. This allowed me to increase my FPS considerably.