Paul Pruvost
Clément Dekeister

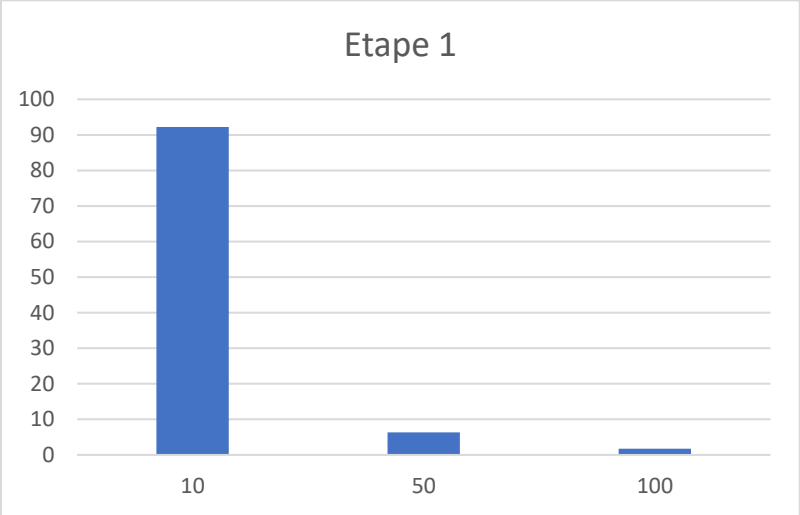
SAE 1.02

Etape 1 :

Pour cette étape, le problème venait de la mise à jour de l'affichage qui se fait pour un personnage à la fois. Le changement était à faire dans le programme pandemic.py où il fallait décrémenter la mise à jour du jeu :

```
    # display update
    deltaTime = clock.tick()
pygame.display.update()
frameNumber += 1
    # time.sleep(0.2)
```

Graphique de performances sur une moyenne de 5 tests pour 3 populations différentes :



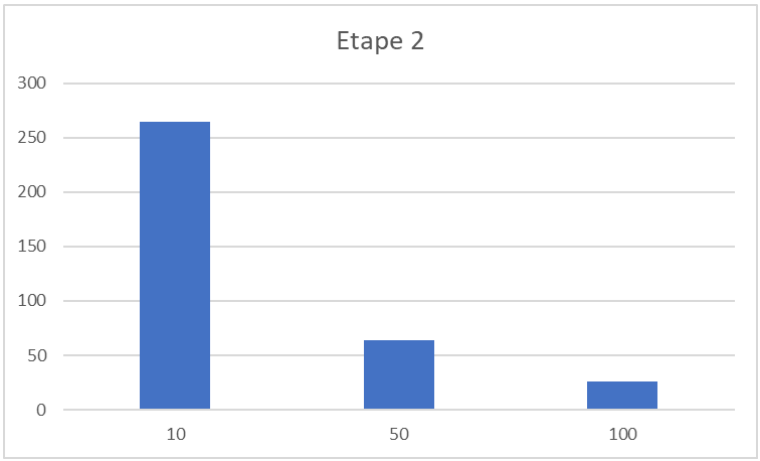
Paul Pruvost
Clément Dekeister

Etape 2 :

Pour l'étape 2, nous sommes passés par un détournement du calcul, au lieu de passer par un calcul de points, nous sommes passés par un calcul de distance (mis en commentaire pour l'étape 3):

```
def circleCollision(c1, c2):

    d2 = int((c1[0]-c2[0])*(c1[0]-c2[0]) + (c1[1]-c2[1])*(c1[1]-c2[1]))
    if d2 < (constants.PERSON_RADIUS*2)**2:
        return True
    return False
```

Graphique de performances sur une moyenne de 5 tests pour 3 populations différentes :



Paul Pruvost
Clément Dekeister

Etape 3 :

Pour l'étape 3 il a fallu changer tout le code. Afin d'obtenir le résultat le plus optimisé nous avons fait 2 versions qui même si elle avait le même fond étaient différentes, la première consistait à utiliser pygame pour dessiner un carré et la 2ème n'utilisait pas pygame mais on stockait dans une liste les coordonnées x du côté gauche et du côté droit et également les coordonnées y du haut et du bas Ensuite on compare si les coordonnées du haut du carré2 sont inférieur à celle du côté bas du carré 1 alors la fonction renvoie faux et on fait pareil pour le reste des côtés du carré et si à la fin des conditions le programme n'a toujours pas renvoyé faux alors il renvoie vrai car les carrés se touchent. On a fini par choisir la version sans pygame car elle était légèrement plus efficace en termes de fps

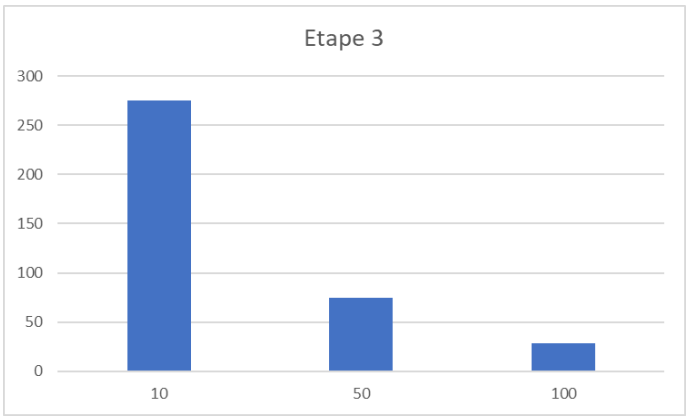
Avec Pygame :

```python
def circleCollision(c1, c2):
    '''
    Determines if two circles intersect or not
    '''
    radius=constants.PERSON_RADIUS
    left1=c1[0]-radius
    top1=c1[1]-radius
    left2=c2[0]-radius
    top2=c2[1]-radius
    width=radius+radius
    height=width
    carre1=pygame.Rect(left1,top1,width,height)
    carre2=pygame.Rect(left2,top2,width,height)

    if carre2.right < carre1.left:
        return False
    if carre2.bottom < carre1.top:
        return False
    if carre2.left > carre1.right:
        return False
    if carre2.top > carre1.bottom:
        return False
    return True
```

Sans Pygame :

```python
def circleCollision(c1, c2):
    '''
    Determines if two circles intersect or not
    '''
    radius=constants.PERSON_RADIUS
    t1=[c1[0]+radius,c1[0]-radius,c1[1]+radius,c1[1]-radius]
    t2=[c2[0]+radius,c2[0]-radius,c2[1]+radius,c2[1]-radius]
    if t2[0] < t1[1]:
        return False
    if t2[1] > t1[0]:
        return False
    if t2[2] < t1[3]:
        return False
    if t2[3] > t1[2]:
        return False
    return True
```

Graphique de performances sur une moyenne de 5 tests pour 3 populations différentes :



Paul Pruvost
Clément Dekeister

Etape 4 :

Pour cette dernière étape, il n'y pas eu tant de modifications à faire, il faut juste rajouter 2 conditions :

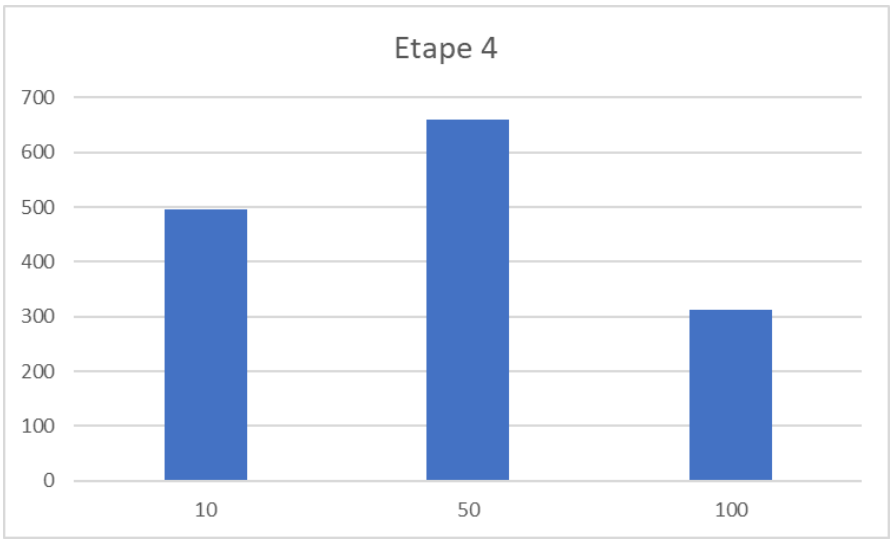
- La première condition se fait dans le programme pandemic dans la fonction constante comme dit dans l'énoncé on n'a pas besoins de savoir si une personne en pleine forme touche une personne en pleine ou une personne guérit donc on rajoute comme condition de lancer le programme de collision seulement si la personne est infectée.

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

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

- Mais ce n'est pas tout car une fois la fonction de collision lancé elle va tester toutes les autres personnes même si elles sont déjà infectées ou guéries alors on ajoute la condition dans le programme engine sur la fonction compute_collision lancer le programme seulement si la personne est en bonne santé :

```
            # draw scene
        draw(scene,font,persons)

            # display update
        deltaTime = clock.tick(30)
```

Graphique de performances sur une moyenne de 5 tests pour 3 populations différentes :



Paul Pruvost
Clément Dekeister

Partie Optionnelle :

Pour avoir le nombre de fps, il suffit de remplir la fonction de pygame qui s'occupe des fps par le chiffre 30 et le tour et joué et il faut aussi sortir cette fonction de la boucle for :

```
            # draw scene
        draw(scene,font,persons)

            # display update
        deltaTime = clock.tick(30)
```

Conclusion :

To conclude we can see on the graphe below that our optmisation work pretty well we went from 0 to more than 300. Even if on the graph it s almost only the last step that upgrade the performance the reason is because the last is very effective but also beacause we didn't think to remove the draw for the loop wich use a lot of memory apparently . The second step was also very useful because the basic function was making like 120 calculations for each comparaison if I remember well wich is just dumb because it's was a lot more complex than just estimate the distance between the center of the two circles . For the upgrade between the 2nd and the 3rd step the difference is not really huge because we almost completely delete all the complex calculations . Lastly to sum up this SAE I would say that we liked this project even if sometimes we spend hours trying to solve one problem but it feel good when we finally find the answer .
Thanks for reading us :)
We hope that you enjoyed our optimisations cause now as we can see on the simulation covid has an end