Avec le système d'exploitation

Appliquer le fond d'écran par programmation

Sous Windows

Sous `Windows`, on peut essayer d'utiliser ces deux réponses sur `StackOverflow` :

- `https://stackoverflow.com/a/1977831` (attention cette réponse n'est pas récente)
- `https://stackoverflow.com/q/65914485` (cette réponse complète la précédente et est assez récente)

Sous GNU/linux avec le bureau GNOME

À l'aide d'un terminal, on peut accéder aux réglages de GNOME, et donc appliquer un fond :

```
$ gsettings set org.gnome.desktop.background picture-uri file:///home/david/mesFonds/fond_cercles.png
```

Sous GNU/linux de manière générale

À l'image de GNOME, beaucoup de bureaux ont une façon particulière de donner la possibilité de changer le fond. Par exemple avec le bureau `Awesome`, la documentation de l'API `https://awesomewm.org/apidoc/popups_and_bars/awful.wallpaper.html`) précise qu'on peut le faire en langage Lua avec la commande `awful.wallpaper`.

Néanmoins, on peut utiliser un programme spécifique ne dépendant pas du bureau, par exemple `hsetroot` qu'il faudra installer au préalable.
Par exemple sous Debian :

```
$ sudo apt install hsetroot
$ hsetroot -center /home/david/mesFonds/fond_cercles.png
```

Changer le fond d'écran par programmation (sous GNU/linux)

L'idée est d'exécuter de manière répétitive, à intervalle de temps régulier, notre script de fond de cercles puis de l'appliquer. Les systèmes d'exploitation ont tous un mécanisme qui permet d'appeler à fréquence régulière un programme.
Sous GNU/linux, l'utilitaire `cron` permet de faire cela :

1. Installation : `$ sudo apt install cron`
2. Création d'un fichier en langage `bash` qui appelle le script Python puis applique le fond d'écran. On appelle ce fichier `changeFond.sh` qu'on met dans notre `home` :

   ```
   #!/bin/bash
   python3 /home/david/mes_scripts/monfond.py
   hsetroot -center /home/david/mes_scripts/fond_cercles.png
   ```

   Précisons que ce script ET l'ensemble des fichiers crées dans l'activité doivent alors se trouver dans `/home/david/mes_scripts/`
3. On rend ce fichier exécutable :

   ```
   $ chmod a+x /home/david/changeFond.sh
   ```
4. On ajoute une ligne d'appel régulier dans l'utilitaire `crontab` qui permet de gérer les exécutions à intervalles réguliers :

   ```
   $ crontab -e
   ```

   Un éditeur s'ouvre et on ajoute une ligne à la fin pour exécuter le fichier, par exemple toutes les 10 minutes :

```
19 # 0 5 * * 1 tar -zcf /var/backups/home.tg
20 #
21 # For more information see the manual pag
22 #
23 # m h  dom mon dow   command
24 */10 * * * * /home/david/changeFond.sh
```

Le site `https://crontabkit.com/` peut aider à trouver la bonne syntaxe.

Deux vraies fausses bonnes idées suivantes

- Laisser tourner en continu un programme Python : celui-ci va utiliser beaucoup trop de ressources ;
- faire un fond animé en changeant le fond avec une fréquence trop élevée dans `crontab` : même conséquence que précédemment (en pire).

Pour aller plus loin sur l'image produite

Remplissage des disques

On peut remplir les disques plutôt que de dessiner les cercles. Dans le code de la classe `Cercle`, il suffit d'utiliser `ctx.fill()` à la place de `ctx.stroke()`.

Couleurs indexées

Le site `https://htmlcolors.com/palette/2730/tasya` propose une palette de couleurs (ensemble de couleurs qui sont bien associées).

Pour utiliser ces couleurs :

1. Création d'un attribut `palette` dans la classe `Fond`, par défaut il est à `None`, et s'il est affecté, il est de type `list`.
2. On modifie la méthode `tracer_cercles`, on place l'argument `couleur` comme optionnel (avec le blanc par défaut par exemple). Dans le code, on teste si `self.palette` a été utilisé et, le cas échéant, on pioche au hasard dans la palette :

Code Python

```python
if self.palette is not None:
    couleur = random.choice(self.palette)
```

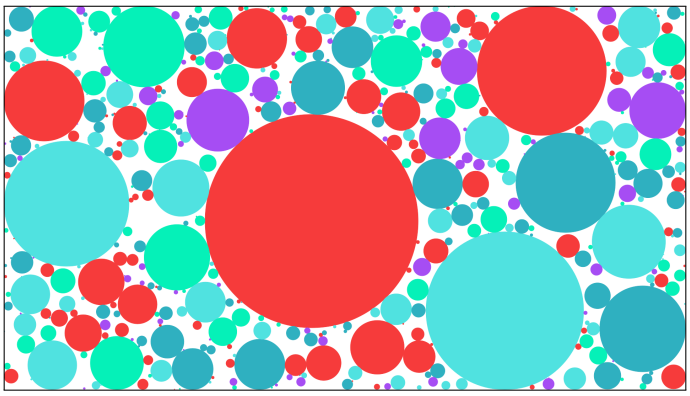
3. On modifie le script `mon_fond.py` :
   - recopie des valeurs du site dans un tableau (de chaînes) ;
   - transformation en couleurs acceptées par `pycairo` grâce au module `colour` (à sans doute installer) ;
   - affectation de la variable `palette` à l'objet fond.

Code Python

```python
import fond_ecran as fe
import colour


mon_fond = fe.Fond()
palette = ['#05F1B8', '#2FB0C0', '#50E2E0', '#A64DF3', '#F63B3B']
palette_rgb = [colour.Color(c).rgb for c in palette]

mon_fond.palette = palette_rgb
mon_fond.tracer_cercles(500)
mon_fond.sauvegarder("fond_cercles.png")
```



Résultat du script précédent.

Appliquer un flou

On peut appliquer un flou au fichier PNG obtenu avec la bibliothèque `PIL(low)`. Contrairement aux imports plus classiques, on importe les modules contenus dans la bibliothèque et qui nous seront utiles : `Image` et `ImageFilter`.

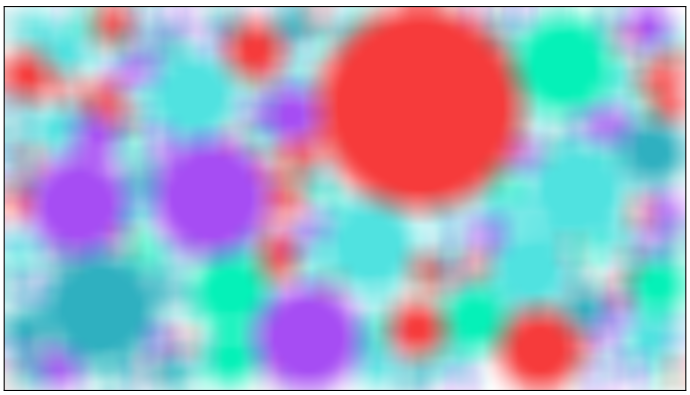
Le principe :

1. Import des modules nécessaires.
2. Ouverture de l'image qui devient un objet.
3. Application dun filtre de flou.
4. Enregistrement de l'image obtenue.

Code Python

```python
from PIL import Image, ImageFilter


im = Image.open("fond_cercles.png")
nouvelle_im = im.filter(ImageFilter.BoxBlur(50))
nouvelle_im.save("fond_cercles_flou.png")
```



Remarque

On pourra consulter la documentation très complète sur `https://pillow.readthedocs.io/`.

Notamment, on notera l'existence du module `ImageDraw` qui permet de dessiner sur une image existante avec un contexte (comme le contexte de `pycairo`) ce que ne permet pas `pycairo`.

Par contre, contrairement à `pycairo`, `PIL(low)` n'est pas adapté pour du dessin vectoriel.

Dessiner des carrés

Plusieurs problèmes sont à gérer :

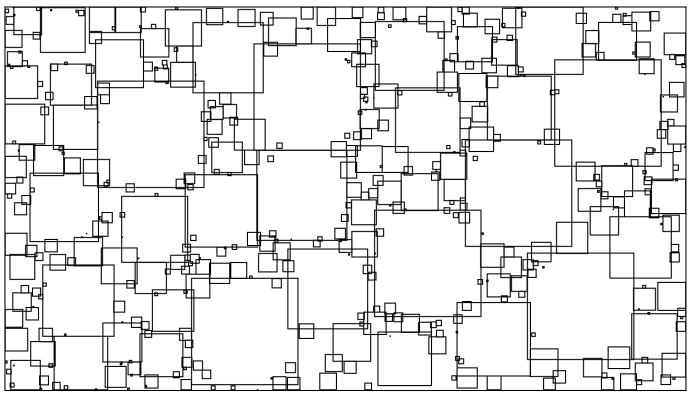
- Écrire une classe `Carre`.
  - Celle-ci prendra en compte, non un rayon, mais une longueur de côté.
  - On conserve la notion de centre de carré (comme centre du cercle). Le tracé se fera alors ainsi :

Code Python

```python
ctx.rectangle(x_centre-self._cote/2, y_centre-self._cote/2, self._cote, self._cote)
```

Les deux premiers arguments sont les coordonnées du point supérieur gauche, ensuite ce sont les dimensions du rectangle ici identiques pour un carré (plus de précision sur la doc. officielle (voir document d'introduction).

- Réécrire ou modifier la classe `Fond` :
  - Réécrire le code du constructeur pour donner le choix de la forme qu'on veut obtenir (cercle ou carré), grâce à un argument optionnel puis un attribut.
  - Renommer les méthodes dont les noms sont liés aux cercles. Le principe reste le même qu'avec des cercles. Le rayon des cercles est simplement remplacé par la moitié du côté du carré.
  - Quand on produit l'image, on observe alors de nombreux chevauchements dans les coins des grands carrés :



Ce problème est liée au simple calcul de distance de la classe `Carre`. Elle n'est pas adaptée aux carrés. Pour cela on change la distance usuelle par le maximum entre les écarts des coordonnées (des deux points) prises deux à deux :

Code Python

```python
# return math.sqrt(sum((b - a) ** 2 for a, b in zip(xy0, xy1)))
return max(abs(b - a) for a, b in zip(xy0, xy1))
```

(C'est ce qu'on appelle la distance associée à la *norme infinie*.)

Le résultat ressemble alors à cette image :