Compte rendu : TD3-DEV5.1

Exercice 1 : Initialisation de git.

1. Nous avons fork le dépo git du meilleur intervenant.
2. Nous avons créer 2 branche.
3. Ensuite localement nous avons `fetch` et `checkout` sur ces branches

Exercice 2 : Programmations

Matthys :

De mon côté je me suis chargé de faire le timer et le système de points sur ma branche `branche_matthy`

Florian:

Pour ma part je pris la charge de la foncionnalité du choix de la difficultés, en prenant un nombre de caractère maximum. et celui ci les mots si la taille le permet le programme choisis plusieurs mots séparé d'un "-" sur la branche `branche_flo`

Exercice 3 : Pull request et Code review

Après que les programmes ont été faits, nous avons créées les pull requests pour passer notre code dans la branche master. Pour ça on passe par l'interface de `gitea` .

Matthys :

Concernant le code de Florian, celui-ci n'avait pas respecté les conventions de nommages (texte en `Français`). Son code était en revanche bien organisé avec des fonctions pas trop longues. Le code étaient également fonctionnel et ne présentait pas de grosse faille de sécurité. Cependant, certains commentaires avaient été oubliés.

Florian

Pour ma part, lors de la review du code de Matthis, celui ci avait respecté quasiment toutes les règles de qualité de code. Les seuls points que je lui ai remarqué avant de pouvoir merge était la taille des ses fonctions qui pouvait dépasser plus de 50 lignes, également quelque commentaires sur le fonctionnement du code qu'il a implémenté.

Exercice 4 : Pull request et Merge

1. Nous avons corrigés nos codes en suivant les commentaires fais par l'autre.
2. Ensuite nous avons `merge` la branche `branche_matthy` dans la master
3. Nous avons `pull` la branche master en local
4. Enfin nous avons `merge` la branche master dans l'autre branche (ici `branche_flo`)
5. Une fois la branche `branche_flo` avec la `master` dedans alors on peut `push` la dite branche dans la master.

Exercice 5 - Complexité

La complexité cyclomatique de la fonction `main` est de 6

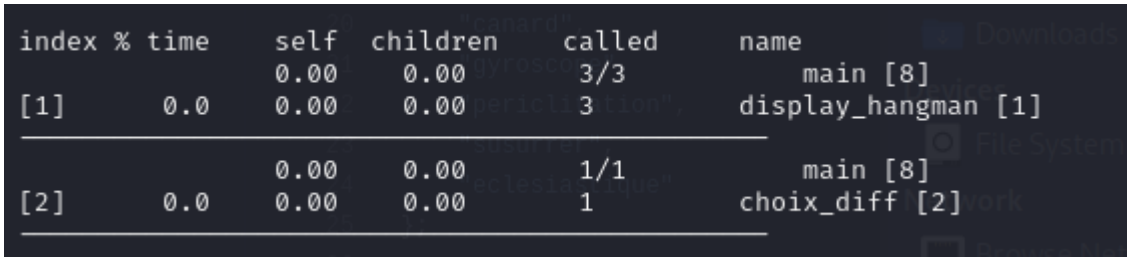
Celle de la fonction `display_hangman` est de 7. (À cause du switch à 7 cas)

Enfin, celle de `choix_diff` à une complexité de : 5

Profilage GProf

```
Each sample counts as 0.01 seconds.
 no time accumulated

  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
  0.00      0.00     0.00        3     0.00     0.00  display_hangman
  0.00      0.00     0.00        1     0.00     0.00  choix_diff
```



```
index % time    self  children    called     name
                0.00    0.00       3/3          main [8]
[1]      0.0    0.00    0.00       3         display_hangman [1]
-----------------------------------------------
                0.00    0.00       1/1          main [8]
[2]      0.0    0.00    0.00       1         choix_diff [2]
-----------------------------------------------
```

On peut voir qu’avec l’utilisation de gprof le temps d’exécution du programme est instantané.

Conclusion générale sur le code

Le code est dans son ensemble propre. La complexité cyclomatique de chaque fonction n'est pas trop elevé.

En terme de ligne, par fonction aucune ne dépasse les 50. Avec donc une complexité basse ainsi qu'une fonction petite il n'y à pas besoin de refractor le code.

Concernant l'efficacité du code, on peut difficilement faire mieux, les quelques boucles `while` que nous avons utilisées sont malgré elles obligatoires.