Projet 2 (semaines 4-7)

Ce projet a pour but de pratiquer à la fois comment travailler avec Git dans une équipe de développement et comment réaliser le re-factoring pour maintenir la qualité du code au cours de la vie du projet/produit.

Pour GIT : *Il ne nous faut pas forcément de connaître toutes les commandes de GIT mais il faut surtout connaître les concepts et savoir à quel moment il faut exécuter une action précise.*

Pour le re-factoring*:* *il nous faut savoir comment améliorer la qualité du code existante, savoir comment utiliser les outils faciliter ce processus.*

Dans ce projet, le code complet du jeu est donné. Vous le trouvez dans le fichier ***airplan_war.pdf.*** **Cependant c'est le code de basse qualité, il vous faut l'améliorer**.

1. Contexte du projet

Vous êtes fourni un repo avec le code base d'un jeu de battre. Ce code base n'est pas complet et il y a de problèmes de la qualité. Vous devez compléter le code et améliorer la qualité du code dans les étapes suivantes.

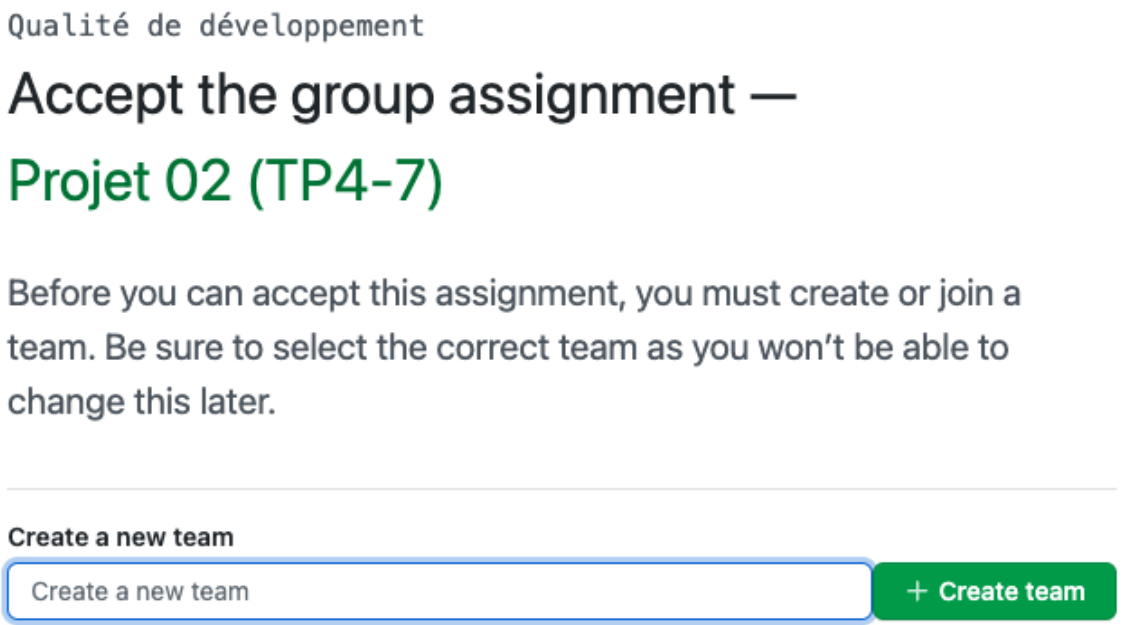
Dans ce projet, vous travailler dans une équipe de 3 personnes : 2 développeurs (un binôme d'étudiants) et un chef d'équipe (l'enseignant). Il vous faut identifier au début qui est le développeur numéro 1 et numéro 2.

Ce TP est noté, correspondant à 35% de la note finale. Il est noté basant sur tous les commits que vous allez réaliser. Alors pour avoir une meilleure note il faut bien attention sur les points suivants :

1. *Le message d'un commit doit être très clair, précis mais il ne faut pas trop long.*
2. *Les commits pendant les séances de TP sont mieux notés. Les commits en dehors de ces heures ont un malus de 20%.*
3. *Avoir le docstring dans tous les fichiers, les classes et les fonctions importants.*
4. *Avoir les commentaires au cas où vous ne pouvez pas expliquer le but par le code soi-même.*

2. Récupérer le repo du projet

Cliquer sur ce lien et suivre les instructions. https://classroom.github.com/a/1bVf3eLZ



3. Début du développement

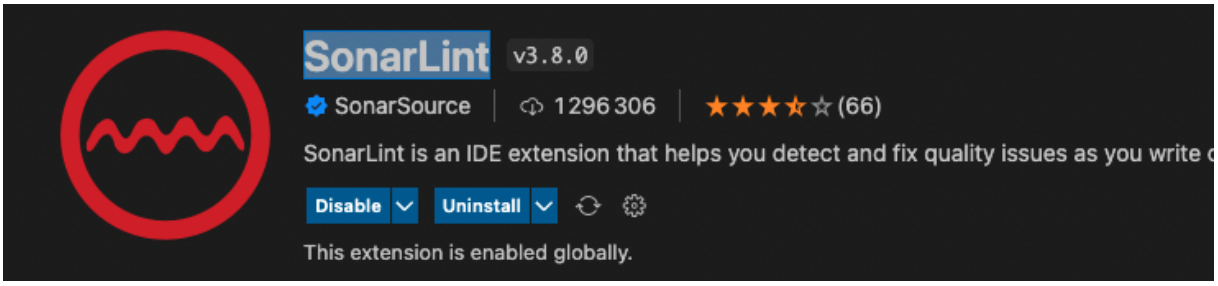
Cloner le repo distant dans votre machine.

Il est recommandé d'utiliser Visual Studio Code pour éditer le code.

Le développeur 01 va mettre à jour le fichier *readme.md* pour ajouter vos informations, et vos numéros de développeur, directement dans la branche *main* et commiter des modifications au dépôt distant.

A partir de maintenant, quand on dit commiter des modifications on comprend que **ces commits sont aussi bien envoyés au dépôt distant.**

4. Installer plugin SonarLint pour VSC (ou votre IDE)



5. Utiliser la méthode Git-Flow

a. Le Git-Flow est présenté dans les slides du cours et dans le tutoriel

i. https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

b. Le développeur 2 crée la branche *develop* et commiter au dépôt distant

6. **Ajouter la fonctionnalité « Faire l'avion se déplacer »**

   a. L'enseignant crée un « github issue » pour cette fonctionnalité et assigner au développeur 2.

   b. Développeur 2 travaille sur cette issue

      i. *git pull -all & git checkout develop*

      ii. *git checkout -b move_plan*

      iii. Implémenter la fonctionnalité bouger l'avion

      iv. Committer les modifications

      v. Créer un PR de cette branche *move_plan* à la branche *develop*

         1. *Il faut mettre une description expliquant ce que vous avez fait, y compris l'amélioration de la qualité du code.*

         2. *Assigner l'enseignant comme le Reviewer*

   c. Révision, feedback, merge ce PR par l'enseignant

      i. Le développeur 2 doit corriger les problèmes basant sur le retour de l'enseignant

      ii. L'enseignant va merger ce PR si tout est ok

7. **Ajouter la fonctionnalité « Dessiner les avions ennemis et les faire avancer vers notre avion »**

   a. L'enseignant crée un « github issue » pour cette fonctionnalité et assigner pour le développeur 1

   b. Le développeur 1 travaille sur cette issue

      i. *git checkout develop & git pull origin develop*

      ii. *git checkout -b enemy_plans*

      iii. Implémenter la fonctionnalité

      iv. Committer les modifications

      v. Résoudre les conflits s'ils existent

         1. *git fetch*

         2. *git switch develop*

         3. *git switch enemy_plans & git merge develop*

         4. Corriger les conflicts

         5. Committer les modifications

vi. Créer un PR de cette branche *enemy_plans* à la branche *develop*

1. *Il faut mettre une description expliquant ce que vous avez fait, y compris l'amélioration de la qualité du code.*
2. *Assigner l'enseignant comme le Reviewer*

c. Révision, feedback, merge ce PR par l'enseignant

i. Le développeur doit corriger les problèmes basant sur le retour de l'enseignant

8. Ajouter la fonctionnalité « Dessiner les puces (bullets) envoyés par notre avion »

a. Le développeur 2 crée un « github issue » pour cette fonctionnalité et assigner pour le développeur 1

b. Le développeur 1 travaille sur cette issue

c. Créer le PR, assigner le développeur 2 comme reviewer

d. Révision, feedback, merge ce PR par le développeur 2

i. Pour ce REVIEW, communiquez par GITHUB, ne questionnez pas et ne faites pas de commentaires directement (même si vous êtes côte à côte)

9. Ajouter la fonctionnalité « Avion ennemies exploser si touchés par une puce »

a. Le développeur 1 crée un « github issue » pour cette fonctionnalité et assigner pour le développeur 2

b. Le développeur 2 travaille sur cette issue

c. Créer le PR, assigner le développeur 1 comme reviewer

d. Révision, feedback, merge ce PR par le développeur 1

i. Pour ce REVIEW, communiquez par GITHUB, ne questionnez pas et ne faites pas de commentaires directement (même si vous êtes côte à côte)

10. Création de la branche de version

a. Les nouvelles fonctionnalités prévues pour la version 1.0 du logiciel se trouvent toutes dans la branche *develop*. Il est donc temps de créer la branche de version qui permettra de tester la prochaine version et d'y intégrer des correctifs avant sa publication. Le développeur 1 réalise cette tâche.

b. git checkout develop

c. git pull

d. git checkout -b release-v1.0

e. git commit --allow-empty -m "Début branche de version 1.0"

f. git push -u origin release-v1.0

g. Une fois la branche créée, le développeur 1 ajouter un commentaire prévenant l'équipe que la branche *release-v1.0* a été créée et qu'il ne recevra que des

correctifs avant d'être intégrée dans la branche *main* pour produire la future version majeure.

11. Vide (ignorer cet étape)

12. Merge a la branche *main* après bien tester

a. Les tests ont été concluants, la branche *release-v1.0* est suffisamment stable pour devenir la nouvelle version stable du logiciel.
b. Les développeurs 1 et 2 ont testé les nouvelles fonctionnalités. Ils ont testé l'installation de la nouvelle version sur différentes distributions Linux ainsi que sur Windows pour être sûr qu'aucun élément n'est dépendant du système
c. Le développeur 1 se charge alors de créer le commit de la version 1.0
    i. *git checkout master*
    ii. *git merge --no-ff release-v1.0 -m " Nouvelle version 1.0"*
    iii. *git push*
    iv. *git tag v1.0*
    v. *git push --tags*
d. Et enfin, le développeur 1 met également à jour la branche *develop* pour qu'elle reçoive les correctifs contenus dans la branche de version :
    i. *git checkout develop*
    ii. *git merge --no-ff release-v1.0 -m "Correctifs v1.0"*
    iii. *git push*
    iv. *git branch -d release-v1.0*
    v. *git push origin :release-v1.0*

13. Ajouter la fonctionnalité « Afficher le nombre de points gagnes »

a. Le développeur 1 crée un « github issue » pour cette fonctionnalité et assigner pour le développeur 2

b. Le développeur 2 travaille sur cette issue

c. Créer le PR et assigner l'autre développeur comme reviewer

d. Révision, feedback, merge ce PR par le développeur 1

    i. Pour ce REVIEW, communiquez par GITHUB, ne questionnez pas et ne faites pas de commentaires directement (même si vous êtes côte à côte)

14. Ajouter la fonctionnalité complémentaire au choix (dev 1)

a. Le développeur 2 crée un « github issue » pour cette fonctionnalité et assigner pour le développeur 1

b. Le développeur 1 travaille sur cette issue

c. Créer le PR et assigner l'autre développeur comme reviewer

d. Révision, feedback, merge ce PR par le développeur 2

    i. Pour ce REVIEW, communiquez par GITHUB, ne questionnez pas et ne faites pas de commentaires directement (même si vous êtes côte à côte)

15. Ajouter la fonctionnalité complémentaire au choix (dev2)

a. Le développeur 1 crée un « github issue » pour cette fonctionnalité et assigner pour le développeur 2

b. Le développeur 2 travaille sur cette issue

c. Créer le PR et assigner l'autre développeur comme reviewer

d. Révision, feedback, merge ce PR par le développeur 1

i. Pour ce REVIEW, communiquez par GITHUB, ne questionnez pas et ne faites pas de commentaires directement (même si vous êtes côte à côte)

16. **Les 2 développeur assurer tous les warnings du SonarLint sont résolus**

a. Révision et merge

17. Création de la branche de version

a. Les nouvelles fonctionnalités prévues pour la version 2.0 du logiciel se trouvent toutes dans la branche *develop*. Il est donc temps de créer la branche de version qui permettra de tester la prochaine version et d'y intégrer des correctifs avant sa publication. Le développeur 2 réalise cette tâche.

b. git checkout develop

c. git pull

d. git checkout -b release-v2.0

e. git commit --allow-empty -m "Début branche de version 2.0"

f. git push -u origin release-v2.0

g. Une fois la branche créée, le développeur 2 ajouter un commentaire prévenant l'équipe que la branche *release-v2.0* a été créée et qu'il ne recevra que des correctifs avant d'être intégrée dans la branche *main* pour produire la future version majeure.

18. Merge a la branche *main* après bien tester