amU Faculté des sciences Aix Marseille Université

Année 2024-25

Domaine Sciences et Technologies
Master informatique
Génie logiciel : TP 7
Codes UE : SINA09EL
Jeu de société Othello

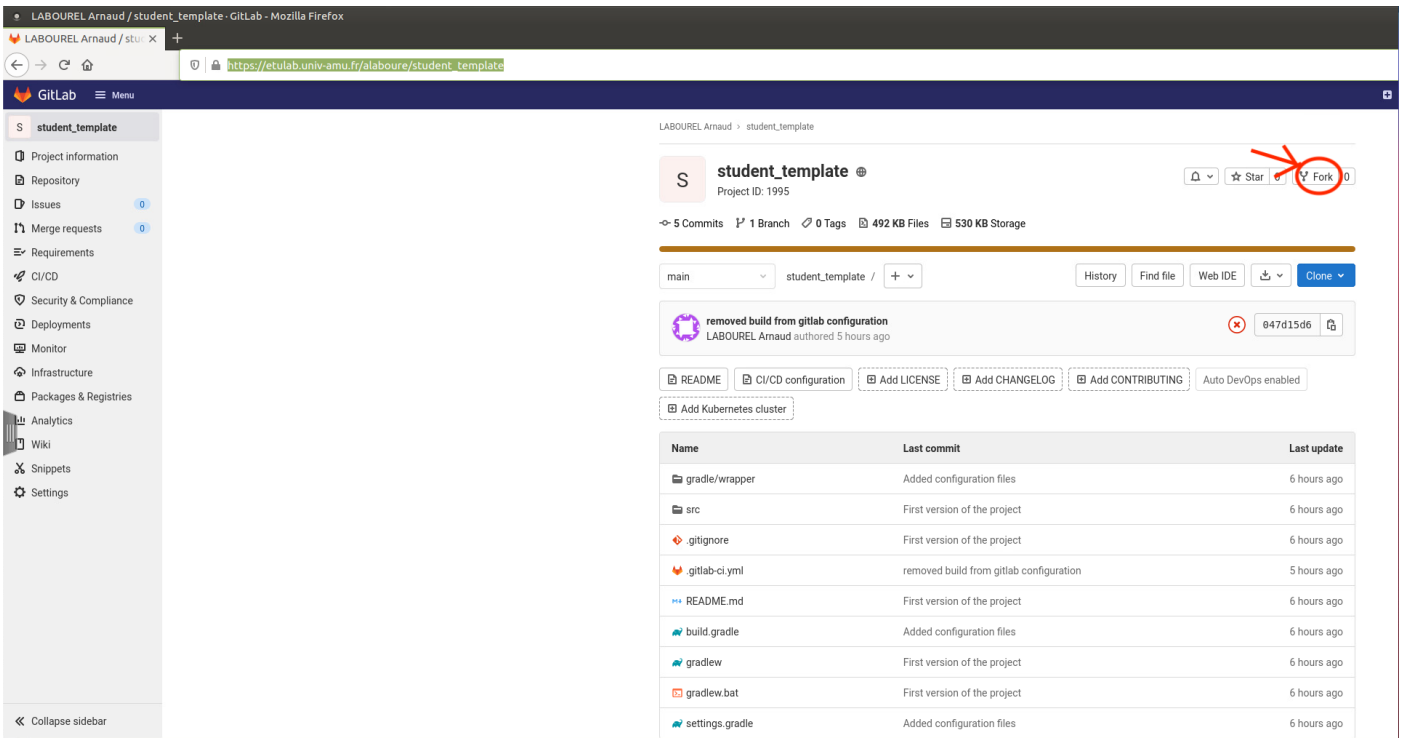
1 Dépôt pour jeux de société

Le but de ce projet est de développer une application graphique permettant de jouer au jeu Othello avec les besoins du client décrit dans la feuille de TD 6.

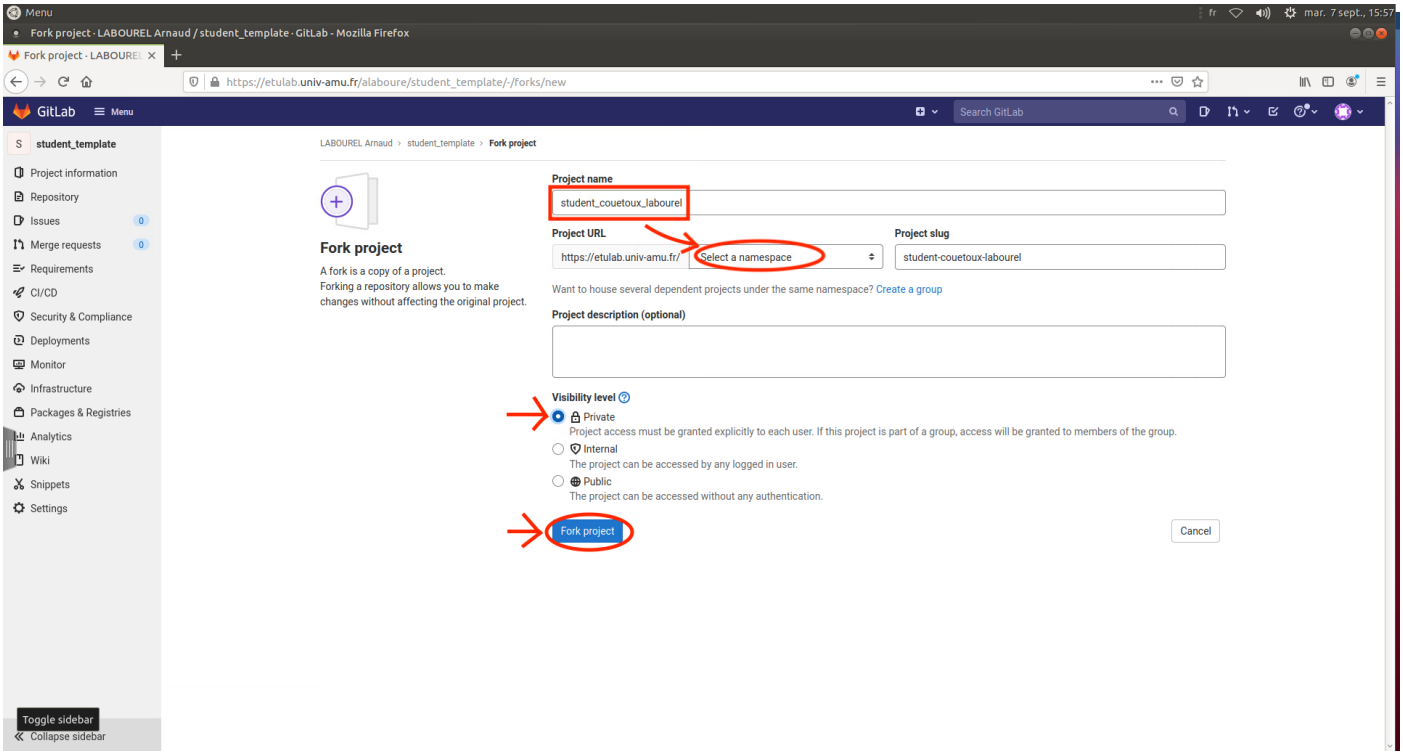
1.1 Fork d'un projet

Vous allez maintenant créer votre projet en utilisant un projet déjà existant. Pour cela, il faut :

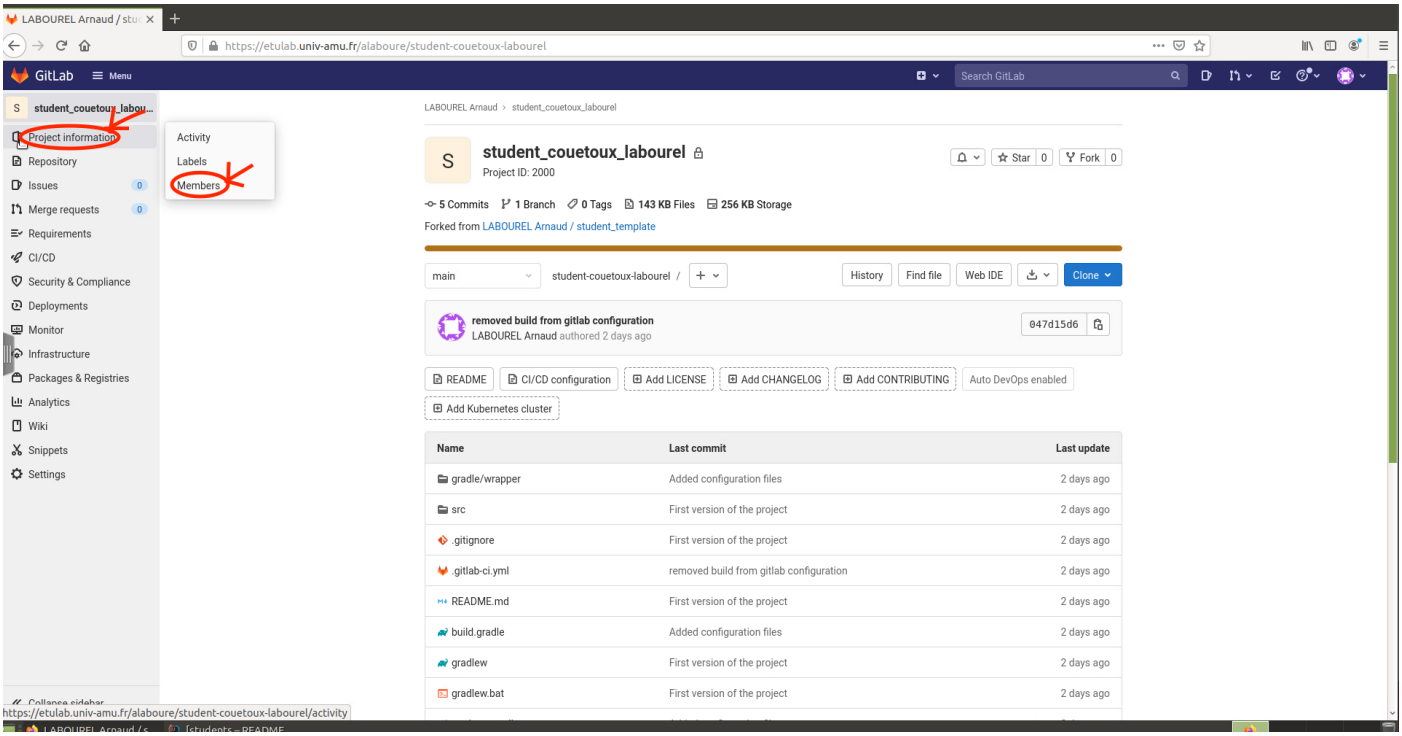
1. Aller sur le projet `board-game-library-template` qui servira de base pour ce TP en accédant à l'adresse suivante : https://etulab.univ-amu.fr/alaboure/board-game-library-template#
2. Cliquer sur le bouton *fork*.



3. Changer le nom du projet pour le changer `othello-game-XX` avec `XX` le numéro de votre équipe. Sélectionner comme espace de nom (*spacename*) votre propre compte. Mettez la visibilité du projet en *private* afin que vos camarades en dehors du projet n'y aient pas accès et validez le fork en cliquant sur le bouton `fork project`.

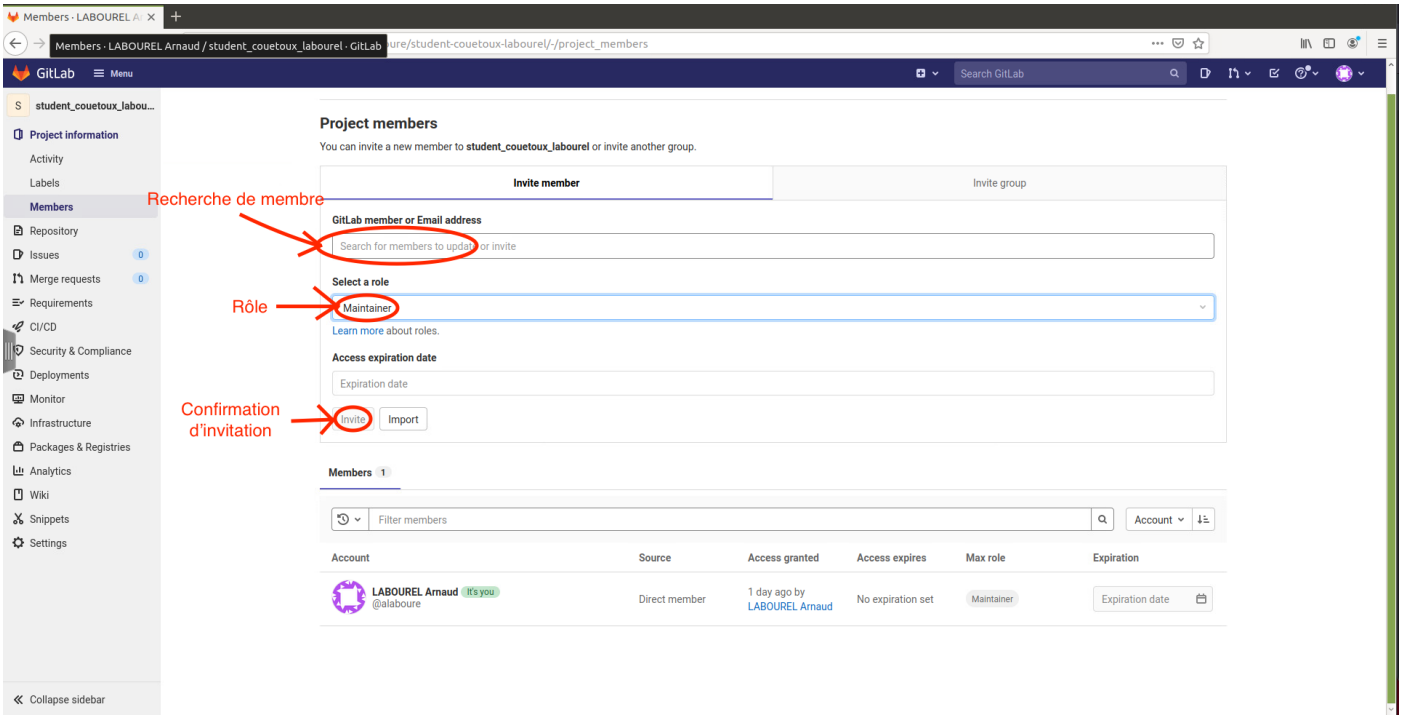


4. Une fois le projet créé, vous pouvez rajouter des membres (par exemple la personne en charge de votre groupe de TP) en cliquant sur `project information` dans le menu de gauche puis `members`.



5. Ensuite vous pouvez rechercher une personne dans la barre dédiée. Une fois celle-ci trouvé vous pouvez

lui donner un rôle (au moins `reporter` pour donner l'accès en lecture du code, `maintainer` pour l'accès en lecture et écriture sur la branche principale *master* ou *main* et *owner* pour donner approximativement les mêmes droits que le créateur du projet), puis confirmer son invitation en cliquant sur le bouton `invite`.



1.2 Explication bibliothèque

Afin de vous faire gagner du temps, le projet contient des classes permettant d'initialiser facilement une interface graphique pour un jeu de société. Le point d'entrée est la classe `JavaFXBoardGameApplicationLauncher` qui permet d'initialiser une interface graphique. Cette classe implémente le patron de conception singleton et donc possède une méthode de classe `getInstance()` permettant d'obtenir l'unique instance de la classe. La classe implémente l'interface `BoardGameApplicationLauncher` qui contient l'unique méthode suivante permettant de lancer l'application graphique :

```
void launchApplication(BoardGameConfiguration configuration,
                       BoardGameController controller);
```

Pour lancer l'application il faut donc récupérer une instance de `JavaFXBoardGameApplicationLauncher` et lancer la méthode `launchApplication` avec :

- Un `record` BoardGameConfiguration qui va donner
  - Le nom `String title` de l'application ;
  - Les dimensions `BoardGameDimensions dimensions` du plateau de jeu ;
  - La configuration des éléments `List<LabeledElementConfiguration> labeledElementConfigurations` formant la barre de menu. Chaque élément sera défini par

un `LabeledElementConfiguration` qui défini un texte de l'élément `String label`, un identifiant `String id` et un type d'élément `LabeledElementKind` qui peut être bouton (`BUTTON`) ou texte (`TEXT`).

- Un contrôleur `BoardGameController controller` qui implémente les trois méthodes suivantes :
  - `void boardActionOnClick(int row, int column)` qui est la méthode appelée quand la case (`row`, `column`) est cliqué ;
  - `void buttonActionOnClick(String buttonId)` qui est la méthode appelée quand le bouton d'identifiant `buttonId` est cliqué ;
  - `void initializeViewOnStart(BoardGameView view)` qui est la méthode appelée au démarrage de l'application qui initialise la vue avant l'affichage de l'interface. L'interface `BoardGameView` donne les méthodes suivantes pour modifier la vue :
    - `void addShapeAtSquare(int row, int column, Shape shape, Color color)` qui ajoute une forme `Shape` de couleur `Color` à la case (`row`, `column`) ;
    - `void removeShapesAtCell(int row, int column)` qui retire toutes les formes présentes dans la case (`row`, `column`) ;
    - `void setCellColor(int row, int column, Color color)` qui fixe la couleur de fond de la case (`row`, `column`) ;
    - `void updateLabeledElement(String id, String newText)` qui met à jour un texte d'un bouton ou texte d'identifiant `id`.

Vous trouverez davantage de détails dans la documentation des deux `packages` suivants :

- javadoc `package` fr.univ_amu.m1info.board_game_library.graphics
- javadoc `package` fr.univ_amu.m1info.board_game_library.graphics.configuration

Le projet contient un exemple d'utilisation de ces classes sous la forme d'une classe `HelloApplication` dans le package `fr.univ_amu.m1info.board_game_library` qui peut être lancée via la tâche `gradle run` du projet.

2 Critères d'évaluation

Vous serez évalué sur :

- **La conception logicielle :** votre projet devra dans la mesure du possible respecter les bonnes pratiques de conception logicielle en programmation orientée objet tels que les principes SOLID. Par exemple, des classes ayant trop de responsabilités vous pénaliseront.
- **La propreté du code :** comme indiqué dans le cours, il est important de programmer proprement. Des répétitions de code trop visibles, des noms mal choisis ou des fonctions ayant beaucoup de lignes de code (plus de dix) vous pénaliseront. Le sujet vous donne les méthodes que vous devez absolument écrire, mais il est tout à fait autoriser d'écrire des méthodes supplémentaires, de créer des constantes, ... pour augmenter la lisibilité du code. On rappelle que vous devez écrire le code en anglais.
- **La correction du code :** on s'attend à ce que votre code soit correct, c'est-à-dire qu'il respecte les spécifications dans le sujet. Comme indiqué dans le sujet, vous devez tester votre code pour vérifier son comportement.
- **Les commit/push effectués :** il vous faudra travailler en continu avec `git` et faire des *push/commit* le

plus régulièrement possible. Un projet ayant très peu de *push/commit* effectués juste avant la date limite sera considéré comme suspicieux et noté en conséquence. Un minimum accepté pour le projet sera d'au moins **1 push par sprint**. Chacun des membres du projet devra réaliser au moins un *push* sur la durée du projet.

— **La qualité de rédaction des artefacts scrum :** vous devez faire attention à la rédaction des différents documents dont on vous demande le rendu pour chaque sprint (Sprint Backlog, Product Backlog, Rapport). La qualité de rédaction () et le respect du format qui vous ont été donné fera partie des critères d'évaluation.

3 Tâches à accomplir

Pour chacun des sprints, ce sera à vous de définir les *user stories* sur lesquels travailler et les tâches associées. Une tâche qui devra forcément apparaître pour le premier sprint sera la familiarisation avec la bibliothèque graphique donnée dans le dépôt.

4 Consignes pour le rendu

4.1 Planning des sprints

— sprint 0 : 25 octobre-7 novembre
— sprint 1 : 8 novembre-21 novembre
— sprint 2 : 22 novembre-5 décembre
— sprint 3 : 6 décembre-12 décembre

4.2 Rendu de chaque sprint

À la fin de chaque sprint Y, on vous demande de rendre les documents suivants :

— les versions finales des documents finaux du sprint N (Sprint Backlog, Product Backlog, Rapport)
— les versions préliminaires (Sprint Backlog) du Sprint suivant (numéro $N + 1$).

Le rendu devra être fait via le lien de rendu de votre groupe de TD :

— Rendu groupe 1
— Rendu groupe 2
— Rendu groupe 3

Les fichiers devront être regroupés sous une archive ZIP qui devra avoir le nom suivant : `equipe-XX-sprint-Y.zip` avec `XX` le numéro attribué à votre équipe et `N` le numéro du sprint.