

UE15 – Informatique appliquée

Projet UML/Sqlite/C++

HelHa
Pluquet Frédéric
2020-2021

On vous demande de vous créer une application console en C++ qui permette de jouer au jeu de cartes "le 8 américain".

- Les règles du jeu : <https://www.regles-de-jeux.com/regle-du-8-americain/>
- Une démonstration du résultat final du projet avec un joueur humain : <https://youtu.be/IDrz38DJsp8>
- Une démonstration du résultat final du projet sans joueur humain : <https://youtu.be/FgkvpAutkRE>
- Le log de la console pour un jeu uniquement entre 4 joueurs ordinateurs : <https://www.dropbox.com/s/cugj03j38yk2zfp/sortie.txt?dl=0>

Votre application doit ressembler en tous points aux vidéos présentées.

Modalités du projet

Le projet est à faire à maximum deux étudiants. Il sera à rendre maximum pour le 3 janvier à 23h59.

Il y aura un examen sur machine où plusieurs modifications seront demandées afin d'évaluer vos compétences à produire du code seul.




Votre projet sera à déposer étape par étape sur le *repository git* créé pour vous par votre professeur. Merci de n'utiliser que ce repository pour me rendre votre travail. Pour le créer, créez-vous un compte sur gitlab.com avec votre adresse mail HelHa puis envoyez un mail avec vos **usernames** Gitlab à votre professeur.

Votre code doit :

- suivre les principes d'orienté objet : encapsulation, polymorphisme justifié, ...
- comporter des méthodes de maximum 10 lignes
- comporter des noms de classes, méthodes, attributs, paramètres mûrement réfléchis et significatifs.
- faire le moins de recopies d'objets possibles (attention aux recopies automatiques). Par exemple, il ne doit exister qu'un seul objet par carte dans tout votre système. Utilisez donc correctement des pointeurs et des références.
- séparer au maximum le modèle de la vue.

Modalités de remise pour chaque étape

Vous devez organiser votre dossier de la manière suivante :

-  `src` : tous vos fichiers sources, documentés (chaque méthode, chaque classe et chaque fichier)
-  `DiagrammeDeClasses.mdj` : votre diagramme de classes correspondant à cette version (en StarUML)
-  `README.md` : un fichier ReadMe écrit *en markdown* qui contiendra vos noms complets avec vos matricules et qui décrira l'avancement de votre projet (ce qui est fait, reste à faire, ...).

Pour chaque étape, vous mettrez à jour l'ensemble de ces fichiers pour que le tout soit cohérent et *taggerez* cette version du git par le tag "Etape X" (où vous remplacez le X par le numéro de l'étape correspondante).

Attention : il ne faut pas avoir de dossiers Etape 1, Etape 2, ... à la racine mais bien les 3 éléments définis plus haut. Vous committez votre code quand vous le voulez et quand une étape est terminée, vous allez tagger ce dernier commit avec un tag "Etape X". Comme cela, je pourrai suivre facilement votre travail.

Un non respect des consignes sera sanctionné par une perte de points.

Etapes du projet

Pour vous guider dans votre développement, voici les différentes étapes de développement que vous devez suivre.

Etape 1

Nous avons besoin de 56 cartes (As, 2, ..., 10, Valet, Dame, Roi de Pique, Coeur, Carreau et Trèfle) + 4 jokers. Toutes ces cartes sont mises au départ dans la *pioche*.

Créez donc les classes nécessaires pour stocker ces informations et afficher la pioche. Commitez, poussez et taggez.

Etape 2

On va fixer le nombre de joueurs à 4 pour le reste des étapes. Chaque joueur doit avoir un paquet de cartes, qu'on appellera sa *main*. Dès le début du jeu, chaque joueur reçoit 7 cartes depuis la pioche.

Adaptez le code précédent pour stocker ces informations et afficher la pioche et les mains de chaque joueur. Commitez, poussez et taggez.

Etape 3

Nous allons ajouter un *tas* qui va recevoir les différentes cartes jouées par les joueurs. La première carte restante de la pioche est mise sur le tas. Si cette carte est un Joker, on tire une autre carte de la pioche pour la mettre sur le tas (et on recommence jusqu'à avoir une carte autre qu'un Joker sur le tas).

Ensuite, à tour de rôle, chaque joueur va donner une carte de sa main pour aller sur le tas. A cette étape, on ne vérifie pas si une carte peut être posée ou non sur le tas. On la pose simplement et on passe au suivant. On fait cela jusqu'au moment où les joueurs n'ont plus de cartes.

Ici il n'y a pas d'interaction avec l'utilisateur. On prend à chaque tour automatiquement une carte de la main de chaque joueur et on la pose.

Adaptez le code précédent pour stocker ces informations et afficher la pioche, le tas et les mains de chaque joueur à chaque tour. Commitez, poussez et taggez.

Etape 4

Nous allons maintenant vérifier si une carte peut être posée ou non sur le tas. Elle peut l'être si elle a la même valeur (As, 2, ...) ou la même couleur (Pique, ...) que la carte du dessus du tas. Une carte avec la valeur 8 ou un joker peut être posée sur n'importe quelle autre carte.

Si un joker est posé, on garde la même couleur (mais on oublie la valeur) que la dernière carte posée sur le tas qui avait une couleur. Si on a posé un 7 de carreau, puis un joker, on peut alors poser ensuite une carte de carreau, un 8 ou encore un joker.

Ici il n'y a toujours pas d'interaction avec l'utilisateur. On prend à chaque tour automatiquement une carte de la main de chaque joueur qui peut être posée et on la pose, jusqu'au moment où un joueur n'a plus de carte. Si le joueur ne possède pas de telle carte, on passe au suivant.

Adaptez le code précédent pour stocker ces informations et afficher la pioche, le tas, la carte sélectionnée et les mains de chaque joueur à chaque tour. Commitez, poussez et taggez.

Etape 5

Lorsqu'un joueur ne sait pas jouer une carte, il doit prendre une carte dans la pioche. Si il peut la jouer, il la pose sur le tas. Sinon il la garde dans sa main. On passe au joueur suivant.

Si, quand un joueur doit prendre une carte dans la pioche, celle-ci est vide, on la remplit en retirant l'ensemble des cartes du tas.

Adaptez le code précédent pour stocker ces informations et afficher toutes les informations utiles pour comprendre le déroulement de chaque tour. Commitez, poussez et taggez.

Etape 6

Ajoutez les actions des cartes spéciales :

- Les 8 permettent de changer de couleur à n'importe quel moment. L'ordinateur choisit au hasard une couleur quand il pose un 8.
- Les Jokers font piocher 4 cartes au joueur suivant.
- Les Valets font sauter le tour du joueur suivant.
- Les As font changer le sens du jeu.
- Les 2 font piocher 2 cartes au joueur suivant.
- Les 10 font rejouer le même joueur.

Le jeu s'arrête quand un joueur n'a plus de carte.

Adaptez le code précédent pour stocker ces informations et afficher toutes les informations utiles pour comprendre le déroulement de chaque tour. Commitez, poussez et taggez.

Etape 7

Lorsqu'un joueur n'a plus de carte, le round s'arrête et on calcule pour chaque joueur les points qui lui restent dans la main :

- Roi ou Dame : 10 points,
- Valet, As, 2 : 20 points,
- 8 et Joker : 50 points,
- Cartes 3, 4, 5, 6, 7 et 9 : leur propre valeur.

Un nouveau round commence alors et à chaque fin de round, les points de chaque joueur sont accumulés. Dès qu'un joueur obtient au moins 500 points, la partie est terminée et le gagnant est celui qui a accumulé le moins de points durant la partie. A chaque round, le sens de la partie est réinitialisé et le joueur courant est réinitialisé au premier.

Adaptez le code précédent pour stocker ces informations et afficher toutes les informations utiles pour comprendre le déroulement de chaque tour et chaque round. Commitez, poussez et taggez.

Etape 8

Maintenant que l'ordinateur peut jouer correctement au jeu, nous allons introduire le joueur humain.

Pour cela, refactorisez votre code pour que le choix de la carte à poser et le choix de la couleur quand on pose un 8 soient séparés du reste de votre code (deux méthodes bien définies). Utilisez ensuite l'héritage et le polymorphisme pour créer son pendant humain : une interface utilisateur console qui demande d'entrer manuellement les informations quand le joueur est un humain.

Attention : il doit être avoir un maximum de code non dupliqué.

Gérez tous les problèmes liés aux choix de l'utilisateur :

- si on entre des valeurs erronées
- si on ne peut pas jouer la carte choisie
- ...

Considérez ici qu'on n'a que deux joueurs : un joueur humain et un joueur ordinateur.

Adaptez le code précédent pour que tout fonctionne correctement. Commitez, poussez et taggez.

Etape 9

Passez en paramètre à votre `main` le nombre de joueurs humains (entre 0 et 4). S'il n'y a pas de paramètres ou que le paramètre est invalide, on considère alors qu'il y a aucun joueur humain. On décide ici qu'il y aura toujours 4 joueurs. Donc si il y a qu'un joueur humain, il y aura automatiquement 3 joueurs ordinateurs; si il y a deux joueurs humains, il y aura automatiquement 2 joueurs ordinateurs, ...

On doit donc pouvoir lancer votre programme comme cela pour jouer à 3 joueurs humains et 1 joueur ordinateur :

```
./main.exe 3
```

Commitez et poussez.

Etape 10

Depuis vos sources C++, sauvegardez toutes les cartes jouées par chaque joueur dans une base de données sqlite (nommée `parties.db`) et ainsi que les scores finaux de chaque joueur de chaque round et de chaque partie.

Faites un *autre* programme C++ qui affiche :

- tous les scores par ordre croissant sur la date et l'heure du jeu (une seule requête SQL).
- le nombre de parties gagnées par chaque joueur (une seule requête SQL).
- combien de fois chaque carte a été jouée et son taux de jeu (si elle a été jouée 6 fois sur 10 parties, son taux de jeu est de 60%) (une seule requête SQL).

Commitez et poussez. Le projet est terminé !

Attention: L'étape SQL est indispensable à la réussite du projet. Si vous obtenez une note inférieure à la moitié pour cette étape, votre projet ne pourra avoir une note supérieur à 7 et vous devrez donc revenir en seconde session.

Bon travail !