

On vous demande de vous créer une application console en C++ qui permette de jouer au jeu de cartes "6 qui prend !".

- Les règles sont définies ici : <https://www.gigamic.com/files/catalog/products/rules/rules-6quiprend-05-2012.pdf>
- Une démonstration vidéo des règles est disponible ici : <https://www.youtube.com/watch?v=6u5rp96PAqI>
- Une démonstration du résultat final du projet est disponible ici : <https://www.dropbox.com/s/rsopqj8mphqyb41/DemoJeu.mov?dl=0>

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

### **Etape 1**

Implémentez les 104 cartes avec le nombre de têtes de boeufs correspondant. Ajoutez-les dans un paquet de cartes et mélangez-le. Ecrivez un programme qui montre que cela fonctionne. Veillez à bien séparer les codes des classes en .h et .cpp.

Les règles pour attribuer le nombre de têtes de bœufs à chaque carte sont les suivantes :

- la carte 55 a 7 têtes de bœufs
- si la valeur est un multiple de 11, la carte a 5 têtes de bœufs
- si la valeur est un multiple de 10, la carte a 3 têtes de bœufs
- si la valeur est un multiple de 5, la carte a 2 têtes de bœufs
- les autres cartes ont 1 tête de bœuf chacune

*Commitez et poussez.*

### **Etape 2**

Bien que dans le jeu, nous pouvons jouer de 2 à 10 joueurs, nous allons nous restreindre à 4 joueurs. Ajoutez, à votre précédent code, les 4 joueurs et la distribution de 10 cartes à chacun. Ecrivez un programme qui affiche le paquet de chaque joueur et le restant des cartes. Vérifiez bien que tout fonctionne.

*Commitez et poussez.*

### **Etape 3**

Ajoutez les 4 rangées de cartes qui resteront sur la table. Une fois les 10 cartes distribuées à chaque joueur, ajoutez une carte dans chaque rangée. Modifiez le programme pour qu'il affiche le paquet de chaque joueur, les rangées et le restant des cartes. Vérifiez bien que tout fonctionne.

*Commitez et poussez.*

#### **Etape 4**

Implémentez un tour de jeu : chaque joueur choisit une carte dans son paquet, on les classe par ordre croissant sur leur valeur et on les ajoute dans la rangée qui possède la différence minimale entre sa dernière carte et la carte à poser. Si on ne peut pas la poser (car plus petite que toutes les dernières cartes), on ne fait rien pour le moment.

Pour avancer plus rapidement dans le développement, la carte choisie par un joueur est choisie aléatoirement par l'ordinateur (pas d'interaction avec l'utilisateur).

Vérifiez bien que tout fonctionne en faisant beaucoup de tests différents.

*Commitez et poussez.*

#### **Etape 5**

Gérez le cas où une carte ne peut pas être posée car elle est plus petite que toutes les dernières cartes de toutes les rangées. Le joueur possédant cette carte doit alors choisir une rangée. Le nombre de têtes sur les cartes de cette rangée sont alors additionnées et ajoutés au score du joueur (initialement à 0). La rangée est également vidée et la carte du joueur y est ajoutée.

Encore une fois, pour avancer plus rapidement dans le développement, la rangée choisie par un joueur est choisie aléatoirement par l'ordinateur (pas d'interaction avec l'utilisateur).

Modifiez votre programme de tests pour afficher les scores de chaque joueur et vérifiez que tout fonctionne correctement.

*Commitez et poussez.*

#### **Etape 6**

Gérez le cas où le joueur doit poser la 6e carte d'une rangée : le nombre de têtes sur les cartes de cette rangée sont additionnées et ajoutés au score du joueur. La rangée est alors vidée et la carte du joueur y est ajoutée.

Vérifiez que tout fonctionne correctement.

*Commitez et poussez.*

#### **Etape 7**

Implémentez les 10 tours qui forment une manche. A la fin d'une manche, videz les rangées, les paquets des joueurs et redistribuez les cartes comme dans l'étape 3. Redémarrez alors un tour. Vérifiez que le jeu n'est pas terminé après chaque tour (si un joueur a au moins atteint le score de 66).

Vérifiez que tout fonctionne correctement.

*Commitez et poussez.*

## **Etape 8**

Refactorisez votre code pour que le choix automatique d'une carte dans un paquet d'un joueur et d'une rangée soit séparés du reste de votre code (2 méthodes dans une classe). Utilisez 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.

Vérifiez que tout fonctionne correctement. Le jeu doit être fonctionnel.

*Commitez et poussez.*

## **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.

Vérifiez que tout fonctionne correctement.

*Commitez et poussez.*

## **Etape 10**

Depuis vos sources C++, sauvegardez les scores de chaque partie dans une base de données sqlite (nommée `score.db`).

Faites ensuite un petit programme C++ indépendant qui affiche tous les résultats par ordre croissant sur la date et l'heure du jeu. Affichez également (via une requête SQL) le nombre de parties gagnées par chaque joueur.

*Commitez et poussez. Le projet est terminé !*

## **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.

## Modalités de remise pour chaque étape

Votre dossier sera organisé 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
-  `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).

**Bon travail !**