PROJET C++ : WORLD CUP

KOUAKOU Stéphane
N'KOUKA Nathan

Polytech Sorbonne
EISE4

SOMMAIRE

I) Introduction et règles du jeu

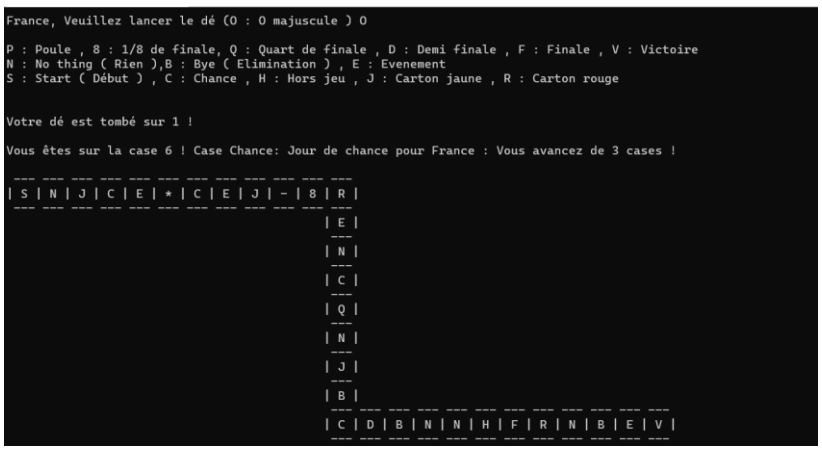
Dans le cadre de notre projet C++ dont le thème porte sur la coupe du monde, nous avons décidé de modéliser un jeu de société version football.

Les règles de ce jeu sont les suivantes :

-Le jeu se joue à 2 joueurs qui jouent chacun à leur tour.
-Chaque joueur choisit un nom et une équipe. Le joueur est libre de choisir n'importe quel nom et n'importe quelle équipe. Cependant, les 2 joueurs doivent avoir un nom et une équipe différente du joueur adverse.
-Le jeu est constitué d'une map contenant de grosses cases (PALIERS) et des petites cases entre les grosses cases (qui représentent des faits de jeu).
- Sur la map, chaque case est représentée par une lettre caractérisant un fait de jeu.
-Le but du jeu est d'arriver sur le palier "V" pour remporter la coupe du monde avant son adversaire en se servant d'un dé (valeur entre 1 et 3) pour avancer. Le dé se lance de manière aléatoire en appuyant sur le "O".
- Le joueur n°1 est représenté par le symbole : * (étoile) et le joueur n°2 par le symbole "-" (trait).
- Il se peut que les 2 joueurs soient sur la même case donc pas de panique si vous ne voyez pas votre symbole, cela veut seulement dire qu'il est caché par le premier joueur sur la case.
-Durant l'avancée sur la map, il faudra faire attention aux nombreuses cases qui vous réservent souvent de mauvaises surprises.

A chaque tour, on vous demandera de jouer, le dé tombera entre 1 et 3, la case sur laquelle vous tomberez sera écrite dans le terminal. Ensuite, la map s'actualisera.

Voici, un exemple de ce qui se passera :



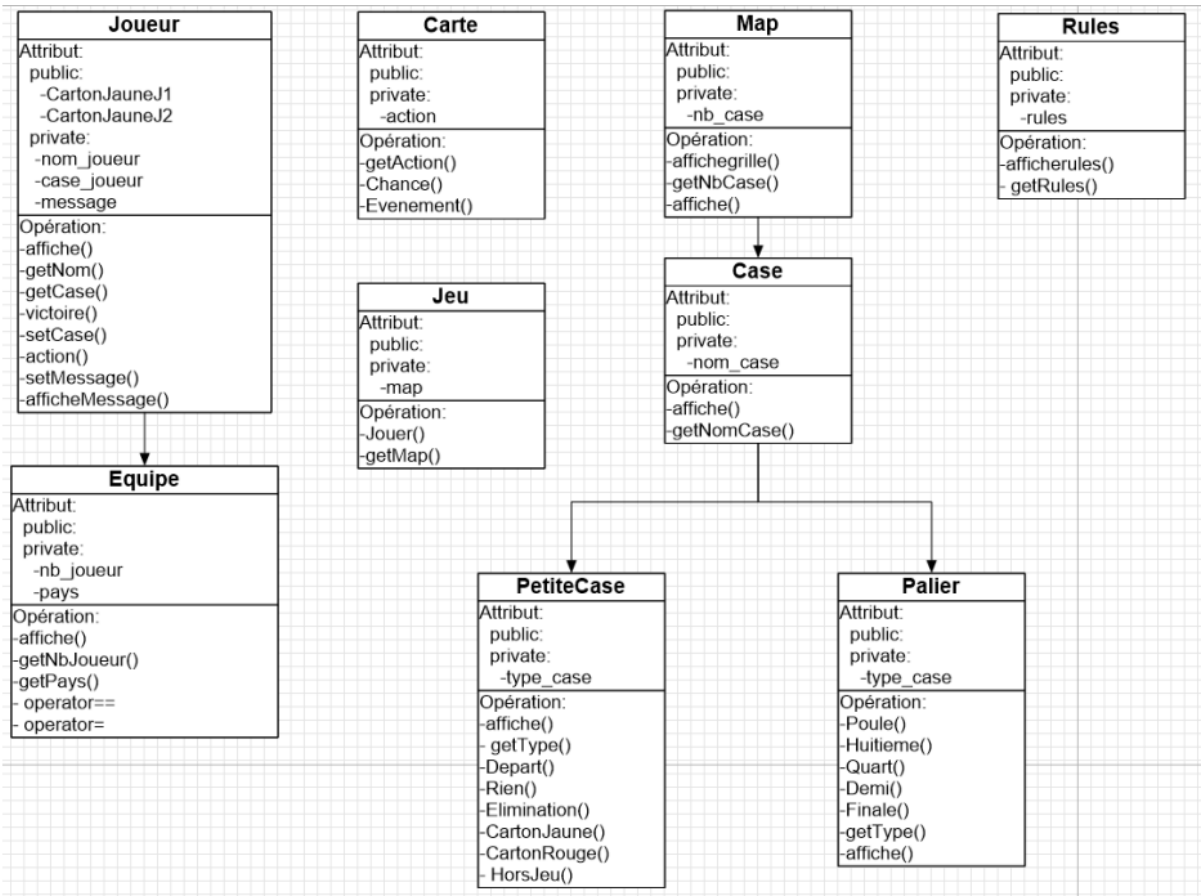
Demande de jouer

Résultat du dé et de la case

Affichage map

II) Diagramme de classe

Notre diagramme se présente comme suit :



Notre digramme de classe respecte les contraintes minimales liées au projet ci-dessous :

- 8 classes ;
- 3 niveaux de hiérarchie ;
- 2 fonctions virtuelles différentes et utilisées à bon escient;
- 2 surcharges d'opérateurs ;
- 2 conteneurs différents de la STL ;
- diagrammme de classe UML complet ;

III) Présentation des fonctions

Parmi les fonctions implémentées, la plus importante est la fonction Jouer de classe “Jeu”.

```
int Jeu :: Jouer(std::list<Joueur> listeJoueur,std::string J1_pays,std::string J2_pays) const
```

En effet, celle-ci va lancer le jeu et c’est dans cette fonction que toutes les autres fonctions du code vont être appelées. Cette fonction va donc gérer l’alternance entre les 2 joueurs pour jouer chacun son tour en se basant sur le retour des fonctions appelées.

Pour l’affichage de la map, la fonction “affichegrille()” va permettre d’afficher la grille du jeu à chaque tour.

```
void Map :: affichegrille(std::map <int,std :: string> map) const
```

Dans celle-ci, on va prendre notre itérateur map et on va afficher par une boucle la valeur clé associée à l’indice i de la map.

Pour la gestion des faits de jeu, on va utiliser la fonction “action” de la classe Joueur.

```
int Joueur :: action(std :: map <int,std :: string> map,Joueur J2,Joueur J1,int cpt){
```

Cette fonction prendra en argument la map actuelle et les 2 joueurs en train de jouer.
Après que l’un des 2 joueurs ait joué, en fonction de la case sur laquelle il est tombé, on va appliquer des faits de jeu. C’est à dire que par exemple, si le joueur est tombé sur une “PetiteCase” alors on va appliquer le fait de jeu associé à la petite case, s’il est tombé sur un Palier, alors on va appliquer le palier associé à cette case et ainsi de suite.

IV) Difficultés rencontrées + respect des contraintes

Parmi les difficultés rencontrées, se trouvent les “setters”.
En effet, au départ, lorsque nous voulions appliquer un setter à un joueur en forçant sa case actuelle à 0, le setter ne s’appliquait pas.

Pourtant lorsque l’on affichait la case actuelle du joueur après avoir effectué un “setter”, cela affichait bien 0. Cependant, lorsque l’on faisait un “getCase()” après avoir effectué un “setCase()” sur le joueur en cours, l’affichage du getter ne changeait pas malgré le setter.

Nous avons résolu ce problème en observant que le setter ne s’appliquait pas dans certains fichiers (.cc) et nous avons dû changer l’emplacement des “setters” dans notre code. Cela a fonctionné par la suite. Cependant, nous n’avons toujours pas compris d’où venait ce problème malgré les heures passées sur ce problème.

Parmi les fiertés de notre code se trouvent d’abord les contraintes imposées qui ont toutes été respectées. En effet, notre code se base sur l’itérateur map afin de créer la map de notre jeu.
Notre map est constituée de 31 cases et chaque case est associée à un fait de jeu représenté par une chaîne de caractère. On a vraiment pu voir l’utilité de l’itérateur map.
De plus,il nous a été très utile pour l’affichage de la map.
En effet l’affichage de la map n’a pas été si facile que ça afin d’avoir un joli rendu sur le terminal et cela en fait une fierté du code.

Nous avons utilisé 2 fonctions “virtual”, des tests unitaires (Test Case). Dans nos tests case , nous avons testé la création de joueurs ,équipe,l’action d’une case et la victoire d’un joueur et nous n’avons remarqué aucune erreur ou de fuites mémoires.

Nous avons utilisé 2 surcharges d’opérateurs afin de vérifier que chaque joueur possède un nom et une équipe différente.

Le nombre de classes et la hiérarchie ont été respectées également. Nous avons entièrement commenté le code afin que la compréhension de notre jeu soit la plus claire possible.

Tous ces efforts nous ont permis de construire un jeu que nous trouvons intéressants, attractifs et compétitif afin que 2 joueurs puissent s’amuser entre eux.

V) Exécution du jeu

Dans l'archive transmise, vous trouverez tous les fichiers (.cc) dont chacun représente une classe. Dans le fichier "allclasses.hh", se trouve toutes les classes de notre projet regroupées dans ce fichier.

Nous avons décidé de faire un fichier (.hh) pour toutes les classes car cela nous évite d'avoir trop de fichier (.hh) et nous évite d'avoir plus de 20 fichiers au total pour notre projet ce qui est beaucoup.
Le dossier "tests" ne sert uniquement pour la réalisation des tests unitaires mais pas pour le projet (on n'y viendra qu'à la fin).

Vous trouverez également un fichier "Makefile". Afin de compiler, aucune librairie n'a besoin d'être installée. Il suffit juste de vous mettre dans le répertoire lié au projet et d'écrire la commande "make" et ensuite le lancer l'exécutable en écrivant : ./projet

Au niveau des tests unitaires, vous trouverez dans le projet un dossier "tests".
Dans celui-ci, se trouve le fichier testcase.cc accompagné des fichiers "catch.hpp" et "makefile".
Dans ce fichier testcase.cc, nous avons testé notre projet selon 3 étapes :

- 1ère étape : Création de joueurs et d'équipes.
- 2ème étape : Création de cases avec effet sur le joueur.
- 3ème étape : Détection de victoire d'un joueur.

Ces 3 tests ont été passés avec succès par notre projet.

Afin de lancer le testcase, il faut juste que vous vous placiez dans le répertoire tests et que vous lancez la commande "make" et ensuite il faut lancer l'éxecutable en écrivant : ./testcase

Si vous avez des soucis ou des questions, n'hésitez pas à nous contacter. Cela serait dommage que vous ne puissiez pas lancer notre projet.

BON MATCH ET BONNE CHANCE AUX FUTURS JOUEURS !