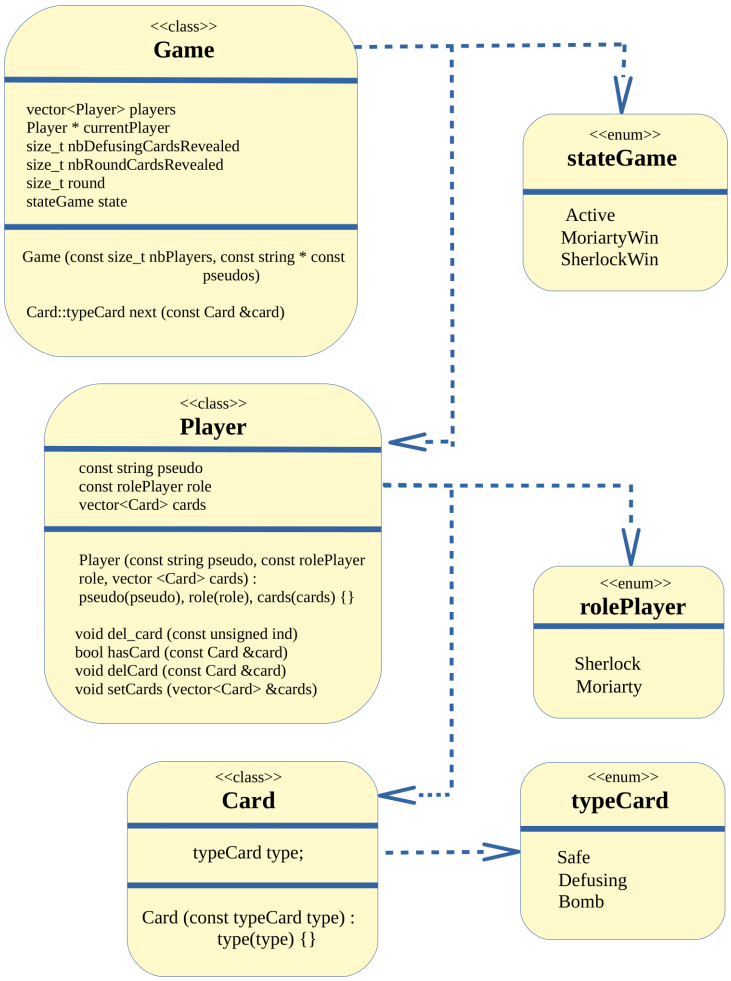
Time Bomb: Deuxième rapport

Ilias Deligiannis, Florent Weber, Nadjib Belaribi, Danyl El-Kabir,
Leonardo Nassabain, François Grabenstaetter

2 novembre 2019

1 État du projet

Dans l'état actuel du projet, nous avons un noyau entièrement fonctionnel ainsi qu'une interface de test en console qui nous permet de facilement vérifier que le jeu fonctionne correctement, et qui est la même interface que celle utilisée par l'interface graphique. L'architecture du noyau ressemble à ceci :



2 Interaction avec le noyau depuis l'extérieur

Chaque module (Interface graphique, Réseau, IA, extensions) pourra communiquer (si implémentés) avec le noyau depuis la classe *Game* (ou par l'intermédiaire d'une classe du module réseau dans le cas d'une partie multijoueurs sur différentes machines).

Nous pouvons distinguer 2 types d'interactions avec le noyau de jeu d'un point de vue du joueur :

- Directement avec les méthodes de la classe *Game*. Cette interaction est destinée au jeu en local sur une seule machine pour tous les joueurs, ou bien pour l'hôte (uniquement) lors d'une partie multijoueurs en ligne avec un hôte et des clients qui s'y connectent.
- Par l'intermédiaire de la classe *GSocket*, qui représente le module réseau de notre programme. Cette classe permet l'interaction des clients avec l'hôte lors d'une partie multijoueur en ligne. En effet, une seule instance de la classe *Game* est nécessaire pour que le jeu puisse se dérouler.

2.1 L'interaction avec la classe Game

Le principe de déroulement du jeu est simple :

1. Instancier un objet de classe *Game* avec en paramètre le *nombre de joueurs* et un tableau de taille *nombre de joueurs* avec leurs pseudo.
2. Dans une boucle, appeler la méthode *next* de la classe *Game* avec en paramètre la carte à dévoiler que le joueur du tour actuel (obtenu avec un appel au getter *getCurrentPlayer*) a choisi.
3. Si le status du jeu est différent de *Active*, cela signifie que soit l'équipe de Moriarty soit celle de Sherlock a gagnée. Il suffit alors de tester cette valeur en fin de boucle avec la méthode *getState*

Tout ce qui est lié à l'affichage de l'état de jeu, au joueur actuel, au nombre de cartes "Defusing" trouvées, au choix de la carte, n'est pas le problème du noyau et doit se faire en fonction de l'état du jeu et des informations accessibles.

2.2 L'interaction avec le réseau (GSocket)

Le module réseau n'étant pas encore entièrement en place, nous ne pouvons pas confirmer les détails d'interaction avec la classe *GSocket*. Néanmoins, nous pensons que les points suivants constitueraient un bon fonctionnement :

1. Les clients, n'ayant pas instancié d'objet de classe *Game*, doivent communiquer avec l'hôte qui héberge la partie *uniquement* via la classe *GSocket*.
2. *GSocket* envoie à chaque client les changements dans l'état du jeu tel que le joueur actuel et la dernière carte dévoilée, le status de jeu, ...

3. Afin d'obtenir plus de détails sur le jeu (nombre de cartes "Defusing" trouvées, numéro de la manche (round), nombre de cartes restantes par joueur, la classe *GSocket* peut implémenter un petit système qui permettra de simuler les variables de jeu et de les modifier selon les changements d'état reçus depuis le réseau (par l'hôte). Cela permettra aux autres modules (partie graphique) de ne pas avoir à s'en préoccuper, et également de ne pas devoir envoyer toutes les données nécessaires par le réseau.

3 Exemple de programme de test du jeu

Le programme que nous avons mis au point afin de vérifier que le comportement du noyau est correct, ainsi que de pouvoir jouer à une version très basique du jeu (pas d'affichage des cartes au début du tour, ni du rôle, ...) possède l'interface suivante :