TP Projet - Sujet A

Christophe Fiorio, Vincent Berry, Elisa Henrion-Gueneau

2021–2022

L'objectif de ce projet est double :

1. vous apprendre à spécifier fonctionnellement une application, ici un jeu
2. vous apprendre à développer une application d'après des spécifications fonctionnelles

Pour cela la classe sera partitionnée en 2 groupements de groupes, appelés A et B.

— les groupes de la partie A devront spécifier le jeu décrit dans le sujet A et implémenter les spécifications fournies par les groupes de la partie B ; pour cela chaque groupe de la partie A sera associé à un groupe de la partie B.

— respectivement, les groupes de la partie B spécifieront le jeu décrit dans le sujet B qui sera implémenté par les groupes de la partie A.

I. Spécification fonctionnelle

Nous présentons ici brièvement le jeu. Une présentation vidéo (`https://youtu.be/-L4-vvIR-f8`) est disponible sur le site de l'éditeur (`https://www.gigamic.com/jeu/quarto`). Vous y trouverez également les règles du jeu.

1 Présentation du « Quarto »

Quarto (voir Figure 1) est un jeu abstrait de réflexion à 2 joueurs inspiré du morpion.

FIGURE 1 – Quarto

Le principe du jeu est le suivant : les seize pièces du jeu, toutes différentes, possèdent chacune 4 caractères distincts : haute ou basse, ronde ou carrée, claire ou foncée, pleine ou creuse. Chacun à son tour choisit une pièce parmi les restantes[1] et la donne à l'adversaire, qui doit la jouer sur une case libre. Le gagnant est celui qui, avec une pièce reçue, crée un alignement de 4 pièces ayant au moins un caractère commun et annonce : « QUARTO ! ».

1. Les pièces n'appartiennent donc à personne.

1.1 Description des éléments de jeu

Le plateau représente 16 positions organisées en 4 lignes et 4 colonnes. (voir Figure 1).

1.2 Mise en place

Le plateau est placé entre les deux joueurs et les 16 pièces du jeu sont mises à disposition des deux joueurs.

1.3 Tour de jeu

Le premier joueur est déterminé au hasard.
À son tour, chacun doit choisir une pièce, la donner à son adversaire pour qu'il la joue. Celui-ci doit la placer sur une des cases du plateau. Si il crée un alignement de 4 pièces partageant une même caractéristique (taille, forme, couleur, densité [2]) il a gagné ! Sinon il choisit à son tour une des pièces restante et la donne au premier joueur.

1.4 Fin de partie

Le premier joueur qui crée un alignement de 4 pièces partageant une même caractéristique a gagné.
Il existe une variante pour les joueurs qui trouvent le jeu trop simple : un carré de 4 pièces adjacentes partageant une même caractéristique est aussi synonyme de victoire. Cela rajoute 9 configurations supplémentaires de victoire.

2 Spécification fonctionnelle

Votre objectif est de fournir au groupe de développeurs les spécifications des fonctions des types abstraits nécessaires de telle sorte qu'une fois implémentées, vous pourrez compiler l'application et l'exécuter.
Les rendus pour cette partie I sont décrits ci-dessous. Il est conseillé de commencer par réfléchir au rendu 2 avant de commencer le rendu 1.

2.1 Rendu 1 : types abstraits

Vous commencerez par définir la spécification fonctionnelle de l'application. Pour cela, vous fournirez différents *fichiers swift*, chacun correspondant à un *type abstrait* dans lequel vous écrirez uniquement le protocole de ce types avec pour chaque fonction en commentaire ce que fait la fonction, le type des paramètres, les éventuelles pre et post conditions et bien sûr la description du résultat.
En aucun cas vous devez fournir une structure de données (struct, class, tableaux, etc...). Exemple, pour le type `IndicatifAeroport` du TP précédent, cela pourrait donner [3] :

Vous ne devez **ni** fournir les structures de données, **ni** les algorithmes des fonctions, juste des `protocol` définissant les types abstraits à utiliser.

2.2 Rendu 2 : programme principal

Vous devez écrire le programme principal de l'application utilisant les types et les fonctions que vous avez définies. Il contiendra l'interface utilisateur.
Bien sûr, vous ne pourrez pas tester ce programme principal puisque vous n'aurez pas codé les fonctions de chacun de vos types. Vous ne pourrez le tester que lorsque les développeurs auront fini d'implémenter vos types.

2. Par densité, on entend plein ou creuse.
3. Vous trouverez sur Moodle un package incluant le fichier protocol et le type `IndicatifAeroport` utilisant ce protocol, ainsi qu'un package l'utilisant.

2.3 Rendu final de la partie I

Vous devrez rendre sous Moodle chacun des 2 rendus conformément aux devoirs demandés.
Puis vous ferez une archive contenant vos trois rendus que vous donnerez à votre groupe de développeurs de l'autre partie de la classe.

II. Description logique et représentation physique

Pour commencer, récupérez la spécification fonctionnelle du groupe de projet qui vous est associé ainsi que le programme principal.

3 Analyse de la spécification fonctionnelle

Vous disposerez d'une séance pour analyser la spécification qui vous est fournie, la comprendre et réfléchir aux structures de données. C'est l'occasion de demander d'éventuels éclaircissements au groupe qui a fait la spécification. Si il vous semble qu'il y a des incohérences ou des manques, c'est le moment de le faire remarquer au groupe spécificateur qui aura alors **une semaine pour corriger son travail**.
Toutes les erreurs de spécification que vous n'aurez pas notifiées à ce moment vous seront imputées.

4 Programmation en Swift

Vous définirez les structures de données de chaque type abstrait et vous écrirez les algorithmes de la spécification fonctionnelle que vous implémenterez.
Ensuite vous testerez le programme principal fourni.

5 Rendu de la partie II

Vous devez rendre sous Moodle une archive contenant tous les types implémentés et le programme principal éventuellement modifié si absolument nécessaire, ces modifications devront alors être justifiées et commentées. Chaque type devra être défini dans un `package` contenant la définition abstraite (le protocol) ainsi que l'implémentation concrète du type (structure de données et code des fonctions). Il est possible de regrouper dans le même package plusieurs types si il y a une logique à le faire.

III. Évaluation

Vous serez évalués dans un premier temps sur votre spécification fonctionnelle. Vous serez pénalisés si l'implémentation de l'autre groupe respecte votre spécification et que le programme principal n'est pas opérationnel.
Vous serez également évalués sur votre implémentation. Celle-ci doit respecter la spécification et si le programme principal ne fonctionne pas, vous devrez indiquer ce qui selon vous ne va pas dans la spécification fonctionnelle qui vous a été fournie.