TP Projet - Sujet B

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://www.youtube.com/watch?v=FCLpHHBHZRI`) est disponible sur le site de l'éditeur (pensez à activer les sous-titres et la traduction automatique) (`https://www.steffen-spiele.de/index.php?id=2399&L=0#popup-4168o`). Vous y trouverez également les règles du jeu en français.

1 Présentation du « Mabula »

Mabula (voir Figure 1) est un jeu abstrait de réflexion à 2 joueurs où le but est de constituer le plus grand groupe de billes adjacentes [1] (composante connexe) de sa couleur.
Le principe du jeu est le suivant : chaque joueur possède 12 billes à sa couleur. Les 24 billes des deux joueurs sont placées aléatoirement au bord d'une grille 6×6.
Chacun à son tour va pousser une de ses billes sur le plateau le long d'une ligne ou d'une colonne, aussi loin qu'il veut, tout en poussant les billes qu'il rencontre sur son chemin. Il n'a pas le droit de sortir une bille du plateau.

1.1 Description des éléments de jeu

Le plateau représente 36 positions organisées en 6 lignes et 6 colonnes. (voir Figure 1).

1.2 Mise en place

Le plateau est placé entre les deux joueurs et les 24 billes sont disposées aléatoirement à chaque bout d'une ligne ou d'une colonne en faisant en sorte qu'il n'y ait jamais plus de deux billes consécutives de la même couleurs, même à travers les coins.

1. Deux billes en diagonales ne sont pas considérées comme adjacentes

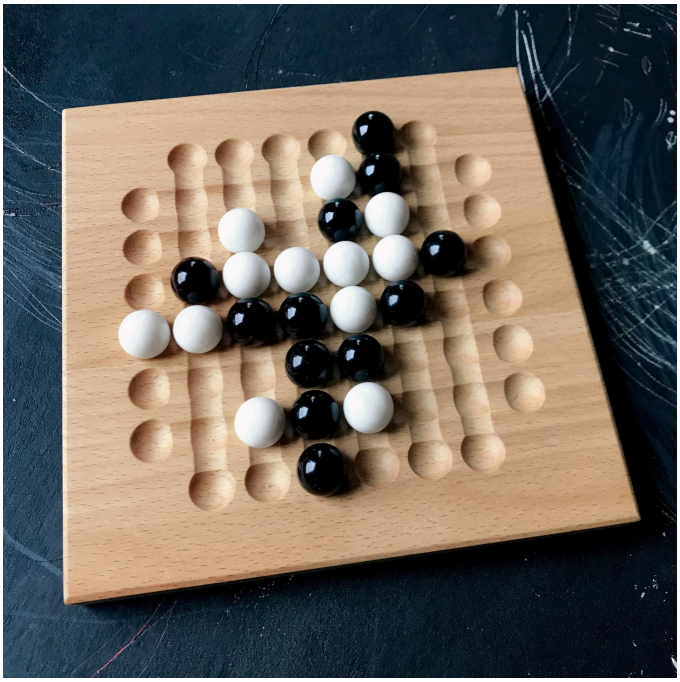

FIGURE 1 – Mabula

1.3 Tour de jeu

Le premier joueur est déterminé au hasard.
À son tour, chaque joueur pousse une bille en ligne droite le long de sa ligne ou de sa colonne en poussant éventuellement les billes rencontrées. Il n'a cependant pas le droit de faire sortir une bille du plateau.
Si un joueur ne peut pas jouer il passe son tour et dès qu'il pourra rejouer, il reprendra la partie

1.4 Fin de partie

Quand aucun des joueurs ne peut jouer, la partie est finie. La plus grande composante connexe a gagnée.
Variante : on multiplie entre elles la taille de chaque composante connexe. Le plus grand score gagne.

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[2] :

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

2. Vous trouverez sur Moodle un package incluant le fichier protocol et le type `IndicatifAeroport` utilisant ce protocol, ainsi qu'un package l'utilisant.

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.

2.3 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.