# DEV4 - Abalone

## Rapport de conception du sous-projet métier

Marika Winska (55047), Oscar Tison (55315)

19 Février 2021

Professeur : R. Absil

Haute École Bruxelles-Brabant

École Supérieure d'Informatique

# 1. Introduction

Ce projet a comme but d'implémenter le jeu Abalone en C++ dans le cadre du cours DEV4. Abalone est un jeu de stratégie où deux joueurs s'affrontent. Chaque joueur commence avec 14 billes. Le premier joueur qui arrive à pousser 6 billes de son adversaire hors du plateau de jeu gagne.

Cette remise contient tous les headers métier du projet avec une courte documentation de ceux-ci. Pour cette remise, le design pattern « Observer / Observable » a été implémenté.

# 2. Classes existantes

Le projet est divisé en 6 classes. Vous pouvez trouver le diagramme de classes en annexe de ce rapport. Ces classes seront expliquées une par une ci-dessous.

## 2.1. Controller

Le Controller est la classe qui contiendra une instance du jeu et contrôlera ce jeu. Cette classe contient une méthode pour commencer le jeu.

### 2.1.1. startGame()

Dans cette méthode, les différentes méthodes du jeu *Game* pourront être appelées. C'est donc le Controller qui va appliquer les mouvements que les joueurs effectuera lors du jeu.

## 2.2. Game

Le classe Game contient l'état d'un jeu. Elle a donc un attribut qui représente le plateau de jeu sur lequel les mouvements seront effectués. Elle garde aussi en mémoire à quel joueur c'est au tour de jouer. Un Game est toujours créé avec un joueur 1 et un joueur 2.

### 2.2.1. checkWon()

Cette méthode contrôle si un des joueurs a gagné. Un joueur a gagné lorsque le joueur adverse n'a plus que 8 billes.

### 2.2.2. makeMove(position posBegin, position posEnd)

Cette méthode permet de faire un mouvement avec une bille. Ce mouvement est uniquement possible si à la position posBegin il y a une bille du joueur

auquel c'est au tour de jouer et si selon les règles on peut bien faire ce mouvement. Tout ceci est contrôlé via les méthodes de la classe Board.

#### 2.2.3. makeMove(position posBeginFirst, position posBeginLast, position posEnd)

Cette méthode fait presque la même chose que la méthode précédente. Dans ce cas-ci on ne bouge pas une bille mais un groupe de billes qui se trouve entre posBeginFirst et posBeginLast (tous les 2 inclus). Ce groupe peut contenir maximum 3 billes qui doivent être du même joueur. Le contrôle pour voir la légalité de ce mouvement est, à nouveau, géré par le Board.

#### 2.2.4. changeTurn()

Cette méthode est appelée quand un joueur a fini son tour. Si c'est au tour du joueur 1, ce sera au tour du joueur 2. Et vice-versa.

### 2.3. Board

Le Board est la classe qui représente le plateau de jeu. Ce plateau de jeu est représenté par un vecteur de pointeurs de billes (voir classe 2.4 Marble). On a donc un vecteur avec les 28 billes. On garde aussi la taille du Board en mémoire pour faciliter sa représentation pendant les prochaines remises.

#### 2.3.1. Constructeur

Dans le constructeur du Board, on prend déjà les 2 Players qui auront été créés dans le Game. De cette manière, on pourra donner ces Players en paramètre des billes qui seront créés dans la classe Board.

#### 2.3.2. marbleAtPosition(position pos)

Cette méthode rendra la bille à une certaine position et null s'il n'y en a pas. On va passer sur tout le vecteur de billes pour trouver la bille à cette position.

#### 2.3.3. isPosPossible(position pos)

Cette méthode contrôle si la position donnée en paramètre est bien possible dans un plateau de jeu de cette taille.

#### 2.3.4. playerAtPosition(position pos)

Cette méthode utilise la méthode marbleAtPosition(position pos) pour après en extraire le joueur à qui appartient cette bille.

#### 2.3.5. deleteAtPos(position pos)

Supprime une bille à une certaine position car celle-ci ne se trouve plus sur le plateau car elle a été poussée en dehors. Avant de supprimer la bille du plateau de jeu, on appelle une méthode de la classe Player pour décrémenter son nombre de billes encore en jeu.

#### 2.3.6. makeMove(position posBegin, position posEnd)

Cette méthode permet de bouger une bille. D'abord, on contrôle, via la méthode isMovePossible(position posBegin, position posEnd), si le mouvement est possible. Ensuite, on change la position de la bille si c'était le cas.

#### 2.3.7. isMovePos(position posBegin, position posEnd)

Contrôle si le mouvement tenté est bien possible. On contrôle donc s'il n'y a pas de bille à la position posEnd et si posEnd est bien dans le tableau.

#### 2.3.8. makeMove(position posBeginFirst, position posBeginEnd ,position posEnd)

Cette méthode permet de bouger un groupe de billes qui se trouve entre posBeginFirst et posBeginLast (tous les 2 inclus). On contrôle, d'abord, si le mouvement est possible via la méthode isMovePossible(position posBeginFirst, position posBeginEnd,position posEnd). Ensuite, on change la position de la bille si c'était le cas. Quand on pousse un groupe de billes on appelle la méthode makeMove mais avec en paramètre le groupe de bille qu'on pousse pour les faire bouger.

#### 2.3.9. isMovePos(position posBeginFirst, position posBeginEnd,position posEnd)

Contrôle si le mouvement tenté est bien possible. On contrôle donc si le groupe de bille qu'on essaie de pousser est bien plus petit que le groupe qu'on bouge. Ou si la position à positionEnd est bien libre.

### 2.4. Marble

La classe Marble représente une bille d'une couleur (blanche ou noire) définie par le joueur donné en paramètre du constructeur.

### 2.5. Position

La classe Position représente une position sur le plateau de jeu d'une bille grâce à trois axes. Le système de coordonnées sur 3 axes renseigné dans les consignes est utilisé pour représenter les positions.

#### 2.5.1. bool isPossiblePos(int size)

Vérifie si une position est possible en se basant sur la taille du plateau donné en paramètre. La méthode retourne true si la position se trouve dans le plateau de jeu, et faux si celle-ci n'est pas possible.

#### 2.5.2. move(int dx, int dy,int dz)

Effectue un mouvement d'une bille par rapport aux 3 axes d'autant d'unité que donné par les paramètres dx et dy et dz. Donc pour une position (0,0,0) sur laquelle on applique la méthode move(-1,-1,0) on arrive à la position (-1,-1,0).

### 2.6. Player

La classe Player représente un joueur par un chiffre (1 pour les billes noires, 2 pour les billes blanches).

#### 2.6.1. deleteMarble()

Supprime une bille du compteur initialisé à 14 (le nombre de billes que possède chaque joueur au début de la partie) à un des joueurs lorsqu'une bille ou plusieurs billes du joueur ont été poussée(s) en dehors du plateau de jeu par le joueur adverse.

## 3. Design pattern

### 3.1. Observer / observable

Nous avons implémenté le design pattern « observer / observable ». Le Game sera l'observable. Lorsqu'un mouvement sera effectué avec succès soit le jeu est gagné par un joueur soit on change de tour, le Game avertira ses observers que son état a changé. Ses observers devront à leur tour se mettre à jour. Comme Observer du Game, nous avons choisi la Vue qui affichera le jeu à l'image. De ce fait, la Vue sera mise à jour lorsqu'un changement sera effectué dans le Game.

## 4. Conclusion

Pour cette première remise de la partie métier, nous avons décidé d'implémenter le design pattern « observer / observable ». Le design pattern est implémenté avec le Game comme observable, et la vue comme observer. Celui-ci servira à bien afficher l'état du jeu au fur et à mesure que le jeu progresse.

Une version avec les headers de la partie métier est disponible sur git au lien suivant : https://git.esi-bru.be/55047/projet-dev4-55315-55047.git .

## 5. Références


1. *Refactoring, URL : https://refactoring.guru/fr/design-patterns/observer/cpp/example, (consulté le 18/02/2021)*

2. *Stack Overflow, URL : https://stackoverflow.com/questions/318064/how-do-you-declare-an-interface-in-c , (consulté le 17/02/2021)*

3. *Geeks for Geeks, URL : https://www.geeksforgeeks.org/pure-virtual-functions-and-abstract-classes/, (consulté le 19/02/2021)*

4. *Doxygen, URL : https://www.doxygen.nl/manual/docblocks.html, (consulté le 18/02/2021)*


## 6. Annexe