Groupe D112 Matricules :

- 48982
– 49773

Rapport Abalone DEV4 Remise graphique

Table des matières

1 Introduction

Dans le cadre du développement du jeu Abalone pour le cours de DEV4, nous sommes amenés à rédiger un rapport scientifique ayant pour but de justifier les choix de modélisation de la partie métier de l'itération II du développement.

La section suivante portera sur une explication plus détaillée de l'analyse, les choix de modélisation et motivations nous ayant poussé à implémenter un ensemble de classes qui nous serviront lors du développement du jeu dans la partie console.

Dans un premier temps, nous exposerons les caractéristiques principales de nos classes et dans un deuxième temps les relations entre les différentes classes.

Suite à votre remarque lors de la défense de la remise 2, nous avons retiré l'appel de la méthode swapPlayer dans le controller et nous avons ensuite changé la visibilité de la méthode swapPlayer dans la façade. De ce fait, le changement de joueur se fait dans la méthode play qui est public et qui est appelé dans le controller.

2 Analyse

Parmi les énumérations fortement typées, nous avons celles représentant les modes de déplacement d'une bille, la couleur qui s'applique à la fois au joueur et aux billes sur le plateau de jeu. Étant donné que les types de mouvements sont de 2 sortes, il nous a semblé opportun de créer une énumération fortement typée pour cet aspect du jeu. Le même raisonnement nous a poussé à choisir des constantes d'énumération pour les différentes directions également.

L'énumération Alphabet est utilisée pour représenter la première coordonnée dans l'utilisation du sytème ABA-PRO.

La classe Position représente une position d'un hexagone sur le plateau. Le choix de cette représentation en 3D est justifié par la facilité d'implémentation de ses algorithmes.

Un des points importants est l'implémentation d'un constructeur par recopie car cette classe est centrale au bon fonctionnement des algorithmes qui vont intervenir dans le mouvement des billes sur le plateau.

La classe Board représente le plateau de jeu qui est composée de 61 hexagones. Nous avons aussi une méthode déclarée en privée dans laquelle nous créons nos hexagons et qui sera ensuite appelée dans le constructeur. Ce choix de modélisation s'explique par le fait de ne pas avoir trop lignes de code dans le constructeur.

La méthode neighbour permettra de savoir si l'hexagone à un voisin vers une direction donnée.

Nous avons utilisé des pointeurs intelligents dans plusieurs classes du modèle notamment dans la classe Board car on voulait éviter des fuites mémoires.

Pour pouvoir utiliser le sytème ABA-PRO en 2D, nous avons implémenté la méthode translate2D qui grâce à une lettre et une valeur retourne la coordonnée sur le plateau de jeu.

La classe Square est utilisée pour représenter les coordonnées dans un système à 2D.

La classe Player qui représente un joueur est simple dans son aspect car la couleur suffit à distinguer les protagonistes.

La classe Marble qui représente une bille possède un unique attribut qui est sa couleur, est associé à un objet de la classe Hexagone qui permet de le localiser sur le plateau car cette classe possède un attribut position. Par ailleurs, un objet de la classe Hexagone possède un attribut de type optional de la librairie

standard qui permet à cette classe de gérer les situations ou un hexagone ne possède pas de billes associées à sa position.

Enfin nous avons la classe Game qui est la façade du jeu. Elle est dotée des méthodes capable de donner des informations sur l'état de déroulement du jeu dont les méthodes pour connaître le joueur courant, si le jeu est terminé etc... Pour éviter que le joueur n'entre n'importe quoi en console, nous avons une multitude de méthodes privées qui vérifiera que les coordonnées entrées par le joueur respectent bien la terminologie ABA-PRO.

La méthode play est la méthode centrale du jeu pour jouer un coup et qui fait appel à plusieurs méthodes privées pour valider le paramètre en entrée et le transmettre le cas échéant aux méthodes distinctes qui gèrent les deux types de coups autorisés dans le jeu.

Dans la partie graphique, nous avons implémenté le design pattern observateur-observer et nous avons implémenté la méthode update qui met à jour l'interface graphique.

Nous avons utilisé un fichier CSS (dans les ressources) dans l'interface graphique pour essayer d'avoir une vue qui ressemble au jeu graphique moderne et d'autre part de séparer les éléments du design visuel du code.

A chaque création de composants, on a veillé à le passer un parent pour que si le parent est détruit, le fils soit aussi détruit automatiquement. Ceci nous permettra d'éviter des fuites mémoires.

3 – Conclusion

L’utilisation de la classe optional dans la partie console a été d’un grand intérêt dans le développement de notre application. Nous estimons en avoir fait un usage judicieux et pas abusif. Bien que la classe game soit la plus grosse classe de notre partie métier mais pour autant elle n’a pas beaucoup de méthodes publiques, la plupart sont des méthodes privées. Pour s’assurer du bon fonctionnement des méthodes privées, nous avons durant tout le développement du jeu mis ses méthodes en publiques et nous avons ensuite testé pour éviter des futures bugs avant de les remettre en privées.

Travailler sur ce projet, nous a permis d’approfondir nos connaissances dans le langage c++ et d’utiliser le framework QT qui est puissant.