Cours : Développement 4

Abalone

Rapport Projet CPP - Bourrel Christopher

1.Introduction

Pour le cours de DEV4, il nous a été demandé de réaliser un projet c++ Abalone, un petit jeu qui se joue à 2 joueurs. Ce jeu devait être remis en 3 parties :

A. La remise 1 du 19 février 2021 qui concerne la partie analyse. Elle nous a permis de créer nos plans pour le développement du projet.
B. La remise 2 du 2 avril 2021 concerne la partie développement console. Dans cette partie j'ai développé tous les mécanismes du jeu et effectué un affichage en console pour y jouer.
C. La remise 3 ... En cours.

1. Règle du jeu

a. Grille hexagonale de 5x5x5 soit 61 hexagones
b. 14 billes de chaque couleur
c. Sans opposition, se déplace dans 6 directions
d. Deux déplacement en *ligne* et en *côte*
e. Déplacement de groupe de bille de 1 à 3 max de même couleur
f. Priorité numérique peut pousser les billes adverses

2. Déroulement de la partie

Le joueur avec les billes blanches commence la partie puis le joueur avec les billes noires joue, une boucle qui continue tant qu' un joueur n'a pas perdu 3 billes.

2.Remise 1 - 19 février 2021

Pour la première remise, j'ai fait une analyse du projet afin de prévoir sa structure générale ainsi que la manière dont il pourra fonctionner. Après mûres réflexions, j'ai prévu mon projet de la manière suivante.

1. Choix du design pattern

La classe Game regroupe toutes les méthodes principales permettant le fonctionnement du jeu. Elle aura un rôle d'interface, les interactions avec les joueurs et le jeu seront possibles grâce aux méthodes fournies par cette classe. La classe view utilisera cette interface afin de proposer aux joueurs une présentation des données fournies par le modèle et une interaction avec elles. Game sera donc un sujet d'observation observé par la classe View, pour cela Game aura comme enfant la classe Observable et View aura comme enfant la classe Observer.

2. Avantages de mon choix

J'ai choisi d'utiliser ce design pattern (observateur observé à l'envers) car le modèle devra fonctionner uniquement au rythme de la vue. Le modèle n'a pas besoin de savoir qu'il est observé par une vue, il devra simplement attendre qu'on appelle ses méthodes pour fonctionner (lors d'une action de l'utilisateur). De plus, cette façon de faire permet à

d’autres développeurs de pouvoir créer leur vue sans prendre en compte le fonctionnement du modèle.

3. Fonctionnement général du jeu

La classe Game sera cadencée au rythme d'une énumération “State” qui donnera au jeu 4 phases principales durant lesquelles les joueurs devront effectuer une action via la vue. En phase de configuration (State::CONFIG), le Game devra configurer la partie, c'est-à-dire, la création du plateau, ajouter les joueurs, placer les billes,etc... En phase de glissement (State:Move) le joueur du tour actuel devra choisir une position initiale et finale à laquelle il poussera sa bille. En phase de mouvement (State::Next Player), le joueur qui jouait donné son tour au prochaine joueur

Les billes (Marble) constituent le jeu. A chaque tour pendant la phase Move, le joueur devra pousser sa bille pour faire sortir les billes de son adversaire.

3.Remise 2 - 2 avril 2021

Cette remise est la plus importante du projet car c’est dans celle-ci que j'ai implémenté toute la logique du jeu et son fonctionnement. J’ai également prévu un affichage en console permettant de voir en action les algorithmes fournis par le cœur du jeu, le modèle.

1. Le modèle (core)

Le modèle a bien été implémenté en suivant globalement ce qui a été prévu dans la nouvelle analyse. Il ne dépend que de lui-même et des infos qu’on lui fournit. Ces informations sont vérifiées avant d’être validées, une de celles-ci est incorrecte, une exception est lancée et le jeu est stoppé. Le modèle n’accepte que certaines informations à un moment déterminé par l’attribut STATE, si le contrôleur appelle une fonction du modèle au mauvais moment, une exception est lancée et le jeu est stoppé.

2. La vue (console)

La vue gérée par la classe View permet d'interagir avec les joueurs en affichant ce qui a été produit par le modèle et en demandant des interactions qui permettent aux joueurs de jouer. En aucun cas la vue ne communique avec le modèle, ses méthodes sont appelées

par l'observer, elle affiche les données fournies et effectue les actions demandées. Elle est donc isolée du modèle.

3. Liste des écarts

Pour mettre au clair, j'ai commencé la remise 2, le 25 mars 2021 suite a aucune nouvelle de mon binôme j'ai du reprendre le projet a 0 avec une nouvelle analyse.

Dans mon projet le mouvement de la bille horizontale n'a pas été implémenté suite au manque de temps.

a. Bug restant

Le jeu fonctionne normalement dans les conditions expliqués, il y a des throw logic_error, qui arrête le jeu, suite au manque de temps je ne me suis pas focaliser sur ça et plus sur le fonctionnement et déroulement du jeu (Bougé,Poussé,Perdre une bille).

b. Problèmes rencontrés

Problème avec les smart pointers. J'ai du choisir de faire des new et avec les problèmes qui suivent, et j'ai eu des problèmes avec mes destructeurs.

c. Justification avertissement

Board.cpp

Ligne 117 : J'ai choisi d'ignorer cet avertissement, cette méthode permet juste de confirmer qu'il est aller dans les méthodes demander pour créer le Game.

Ligne 162 : Changer ou ce trouve le "int i" n'était pas réellement nécessaire vu que je suis dans un for-loop.

Ligne 163 : Changer ou ce trouve le "int i" n'était pas réellement nécessaire vu que je suis dans un for-loop.

d. Estimation du temps sur la remise 2

J'ai passé les 9 jours, à une vitesse de 8 à 10 heures de travail par jour.
Ceci est un cas extrême.

4.Remise 3 - 7 mai 2021