Anas Ben Allal 53203, Julien
Jacobs 52902

Projet : Abalone

Introduction

Le jeux abalone est un jeu de rôle qui a un plateau de jeux hexagonal, il se joue à deux joueur, chaque joueur détient des billes de différente couleur soit noir ou soit blanche. Le but du jeu est d'éliminer les billes adversaires en les poussants en dehors du plateau, le premier joueur à avoir éliminer 6 billes adverse gagne la partie, pour pouvoir éliminer les billes adverses il suffit de pousser ses propres billes afin de mettre hors-jeux la bille adverse.

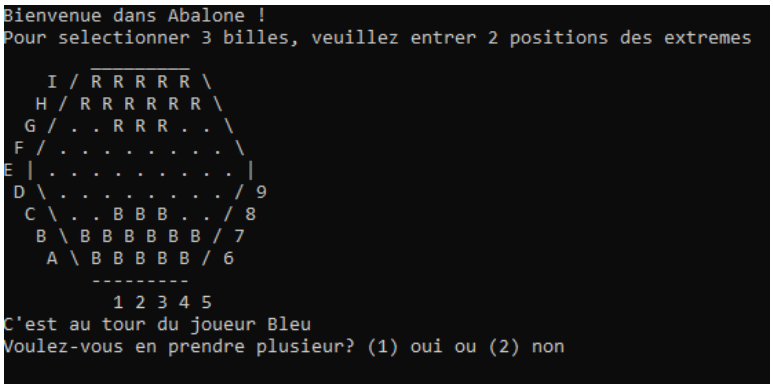
Section métier :

Pour le model, nous avons opté pour MVC (Model View Controller) dans notre Model nous avons comme class : Board , Direction, Game, Piece, PlayerColor et Position.

- **La class Board** qui détient le plateau de notre jeu, nous sommes partis dans l'idée d'un tableau en 2D d'une taille 9x9, nous avons choisis diffèrent méthode : **isInside** pour savoir si la bille est dans notre plateau, **isFree** si la cellule est libre, **swap** qui permet de changer de place les diffèrent bille choisis, **putPiece** pour placer des billes, **push** qui permettra de pousser des billes dans une direction donner, **isFreeAll** permet de savoir si les billes qui pousse vers des billes adverses est supérieur ou inférieur au nombre, et ensuite diffèrent getter comme **getBoard** pour accéder à l'attribut du plateau de type Piece et **getPieceOnPosition** qui donne la bille retourner à une position , **isFreeBoard** pour vérifier si dans la board la case est libre avec comme paramètre les index , une méthode **setPiece** pour changer la pièce reçus en paramètre a des index donner , une méthode **posBoard** pour convertir la position reçu en paramètre en position dans le vector 2D , une méthode **triPiece** pour trié les billes choisis en fonction de la direction recu , une méthode **canPush** pour savoir si les bille peuvent pousser dans une direction donné , **selectDir** une méthode qui permet de savoir dans quel direction les billes se situe entre elle , une méthode **removePieceOnIndex** qui efface la pièce dans les index donné , une méthode **equalsColor** pour savoir si dans une position donné et une pièce donné la pièce a la même couleur ou pas, une autre méthode **removePiece** pour effacer une pièce donner en paramètre , une méthode **colorPos** pour donner la couleur de la pièce a une position donné ,et un Controller pour construire le plateau.

Anas Ben Allal 53203, Julien
Jacobs 52902

Petit aperçu console du plateau :



- **Enum Direction :** les différents choix de direction possible sur le plateau.

HAUT_GAUCHE , HAUT_DROITE, BAS_GAUCHE, BAS_DROITE, GAUCHE, DROITE

- **La class Game** dans cette class nous avons mis 3 attribut privé de type Board , Player et View , une méthode pour sélectionner **select** une bille via la position et sa couleur, une méthode **selectMultiple** pour choisir plusieurs billes, et une méthode **isOver** pour savoir quand le jeu s'arrête ainsi qu'un getter pour l'attribut de la board , une méthode pour un simple mouvement **simpleMove** , un **setCurrent** pour changer le joueur courant , et un getter pour le joueur courant.
- **La class Piece** qui a deux attributs la couleur et la position de la bille, un constructeur pour créer la bille, un setter pour changer la position de la bille a une position donner, un autre setter pour changer la couleur de la pièce , un getter pour la couleur de la pièce et un to_string pour afficher la pièce dans la board.

- **Enum PlayerColor** pour les deux couleurs des billes possible qui est le rouge et le bleu dans notre cas, et également none si la case est inoccupée. **RED** , **BLUE et NONE.**
- **La class Position** comme attribut ou on a les 2 axes en X et Y. Un constructeur position, une méthode **nextPosition** qui renvoie la position dans une direction donné et des getter pour les diffèrent attribut, deux méthode **to_string** pour pouvoir afficher une position et **ostream**

d'injection de flux, un **operator==** , un operateur d'égalité entre deux position.

- **La class Player** comme attribut on a mis color de type PlayerColor pour le joueur qui soit rouge ou bleu, un constructeur et un getter et un to_string pour pouvoir afficher le player.
- **La class Directions :** qui a comme attribut toutes les directions composées en tuple avec la direction et deux nombres pour le déplacement, une méthode **opposite,** qui renvoie la direction opposé donner en paramètre et une méthode **giveDirection** qui retourne la direction donner en paramètre.

Section console :

Dans notre View :

- **La class View** pour afficher en console les différentes méthodes pour interagir avec les joueurs et l'affichage du plateau avec **displayBoard, askPosition, askDirection** , également une méthode qui permet de convertir l'entrer de l'utilisateur **convertChar** afin de convertir les différents alphabet qui compose la board en chiffre pour que la méthode askPosition retourne une position adéquat pour les autres méthodes, une méthode **selectMultiple** qui demande si l'utilisateur souhaite sélectionner plusieurs billes ou bien une seul bille, une methode **messageTurn** pour afficher à qui de jouer , une méthode **messageHelp** pour afficher le début du jeux pour souhaiter la bienvenue et indiquer un message de comment sélectionner 3 bille , une méthode **displayWinner** pour afficher le gagnant de la partie.

Dans notre Controller :

- **La class Controller** nous avons mis deux attribut **View** et **Game**, qu'on initialise dans le constructeur et une méthode **play** pour démarrer le jeu.

Dans notre Main :

- **La class Main :** initialise le Controller et faire appel à play pour démarrer le jeu.

Anas Ben Allal 53203, Julien
Jacobs 52902

Section graphique :

Pour cette partie, nous avons utilisé 3 classes (et également la classe point qu'on a reçu) :

- **Circle** : la class **Circle** permet de créer des cercles pour simuler les billes qui se trouvent dans le plateau, elle est composée d'un constructeur **Circle**, et d'une méthode **paint** qui permet de dessiner sur le plateau.
- **HexaCell** : cette class permet de construire nos hexagones du plateau, en utilisant un constructeur **hexacell** , et de la méthode **paint.**
- **MainWindow** : cette class contient tous les éléments qui permettent de créer l'interface graphique, elle contient un constructeur **MainWindow** , un destructeur , une méthode **handleButton** pour l'interaction du bouton , **displayBoard** pour l'affichage du plateau , une méthode pour récupérer les positions **positionCell** , une méthode pour convertir les char en int **converChar** pour récupérer la position dans le plateau , une méthode pour afficher tous les éléments qui suivent la board les lettres et les chiffres **displayPos** , une méthode pour donner la direction **giveDirection** et une méthode pour afficher le vainqueur **displayWinner**.

Section problème rencontré :

Pour les positions du board , on était partis sur l'idée de donner les index i et j pour chaque position de la board , mais c'est une mauvaise idée car dans le plateau du jeux on peut pas se déplacer comme on le souhaitait , on avait besoin de valeur négatif comme position comme illustré sur l'énoncé du projet.
Au fur à mesure de l'implémentation du projet on s'est rendu compte qui nous manquer des méthodes et quelques un dont finalement on n'avait pas besoin.

Une méthode de tri qui permet de trier la position des pièces que l'utilisateur a choisis en fonction de la direction pour pouvoir déplacer les diffèrent pièces convenablement de sorte que la pièce qui se trouve dans la position 0 du vector de Piece est la pièce qui se trouve en bout de file

dans la direction choisie par l'utilisateur, sans cette méthode la méthode swap ne fonctionne pas comme il le faut. Pour la vue, on a changé car mal organiser pour bien déplacer les pions souhaiter, on est parti sur le même affichage que sur l'énoncé du projet avec les lettres et chiffre pour pouvoir se repérer sur le plateau plus facilement pour bien entrer les bonnes positions par l'utilisateur. L'ajout d'une class Directions, car en cpp on ne peut pas instancier ou donner des valeurs a des énumérations, on y'a mis toutes les directions avec les différentes valeurs correspondant ainsi que deux méthodes une pour donner l'opposer d'une direction donner en paramètre et une autre pour donner la direction donner en paramètre.

En plus de la méthode **push**, on avait besoin d'une autre méthode Boolean (canPush) pour savoir si les pièce reçus en paramètre peuvent pusher dans la direction donner en calculant le nombre de pièce qui se trouve après la première bille dans le vector dans la direction donner.

La méthode **selectDir** qui nous permet de connaitre la direction des billes entre elle pour permettre d'effectuer certain déplacement ou pas , on a créer cette méthode pour pouvoir connaitre l'orientation des 3 billes qui ont était sélectionner , car par exemple si l'utilisateur entre la direction HAUT_GAUCHE et que les 3 billes se trouve en vertical et bien le déplacement peut se faire si il y'a un nombre inférieur de bille dans la direction donner mais si les billes se trouver dans une orientation horizontale et bien le déplacement ne peut pas se faire si il y'a une bille ou plus dans la direction donné.

Une méthode qui retourne la position i et j c'est-à-dire les index pour retrouver la pièce dans le vector pour pas passer au diffèrent position des billes avec des coordonnés parfois négatif.

Pour la partie graphique, au début j'ai construit le plateau rien qu'avec la méthode orbite en partant du centre , je me suis rendu compte par après que la position retourné des différents Hexacell était des positions un peut bizarre , je pense que cela devais être des positions de pixels et de plus il y'avait des hexacell et des circle qui se superposais car la méthode orbite dessinait par-dessus d'autre hexacell et circle , j'ai donc changer mon constructeur hexacell et circle en leur donnant deux variable x et y pour qu'ils puissent me retourner une position voulu , et leur donner

toutes les positions de la board pour pouvoir la construire sans partir du centre du plateau.