Gregory van der Pluijm 54786

Maciej Rolecki 54931

Rapport du projet Abalone

Année scolaire : 2020-2021 DEV4 Professeur : Jonas Beleho

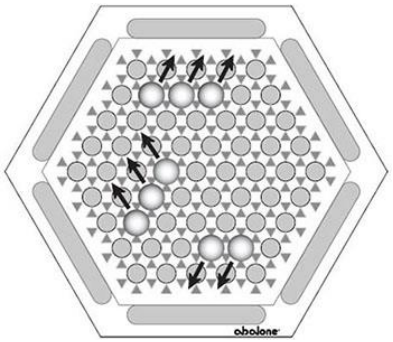
Jeu Abalone, projet de DEV4 codé en C++ par Maciej et Gregory.

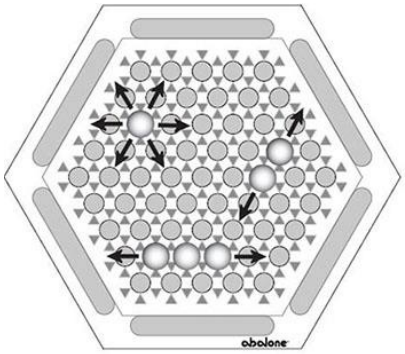
Le projet a été réalisé en deux parties. Nous avons d'abord effectué une version du projet ou l'utilisateur va interagir avec le jeu en tapant ses commandes au clavier, appelée TUI (Terminal User Interface). Par après nous avons effectué une deuxième version où le joueur va interagir avec une interface graphique appelée GUI (Graphical User Interface). En préparant la partie TUI, nous avons préparé la logique du jeu afin que celui-ci se déroule proprement. Cela inclut tous les déplacements possibles par le joueur ainsi que la conversion des coups 'ABAPRO' en coups compréhensible par notre code. Après avoir établi ceux-là, nous avons réalisé une simple interface pour permettre d'afficher le jeu dans la console. Enfin, pour la partie GUI, nous nous somme concentrer à avoir une belle interface graphique mais surtout un lien entre le GUI et la partie logique.

Notre première difficulté durant la réalisation de ce projet fut la représentation d'une balle sur le plateau de jeu ainsi la gestion des coordonnées de celle-ci. Après longues réflexions, nous avons décidé de ne pas avoir d'objet balle mais que chaque case du plateau ait une couleur : NOIRE, BLANC, EMPTY ou OUTOFBOUND. Ceci nous a permis de simplement afficher la couleur de la classe sans avoir besoin d'une liste de balles.
Concernant les positions des balles, nous sommes restés sur un système de coordonnées à 2 dimensions qui nous a permis de ne pas trop compliquer la tâche. L'autre option étant d'utiliser 3 directions qui ne nous est pas quelque chose de familier.

Cela étant, il y a eu la traduction de coups 'ABAPRO' pour les déplacements connus par notre programme. Pour cela, nous avons réalisé la méthode : ABAPRO::getCommand(). Elle accepte un std::string en paramètre. C'est ce string que nous allons analyser pour ensuite en extraire les positions et la direction dans laquelle le joueur veut déplacer ses balles. Pour garder en mémoire toutes les données pertinentes, nous avons créé une struct « MoveUtils ». Elle possède deux attributs pour 2 positions potentielles et une direction qui est la direction dans laquelle les balles doivent bouger.

Ensuite, ce même « MoveUtils » est envoyé à la méthode, ABACORE::applyMove(). Cette méthode va dans un premier temps définir à quel type de mouvement nous avons à faire. On distingue 2 types de mouvements, les mouvements latéraux (2 positions, 1 direction) et les mouvements linéaires (1 positions, 1 direction). Quand la méthode a déterminé quel type de mouvements c'est, la méthode procédera à vérifier si le mouvement désiré est bien valide, si oui, il appliquera le mouvement et affichera les balles dans leurs nouvelles positions.





Finalement, nous avons utilisé le pattern Observateur/Observé pour permettre d'afficher correctement les balles dans la version GUI du jeu. Nous avons la méthode principale qui faire appels à des méthodes secondaires qui vont ajouter toutes les cases et balles nécessaires au fonctionnement du jeu, sur le plateau. Nous avons également une méthode GUI::updateDisplay(), qui va s'occuper de la mise à jour de notre affichage. Dans notre situation, cela correspond à la suppression de toutes les balles du jeu, et du rajout de celle-ci si elles ont été modifiées. Ceci a pu être réalisé avec plus de faciliter grâce à la séparation de la vue en 3 scènes :

- La plus basse/profonde qui correspond juste à l'affichage dans le fond des cases et de leur coloriage en gris. Elle contient également tous les messages affichés à l'utilisateur comme : à qui de jouer, votre mouvement est incorrect.
- La scène du milieu qui détient en elle toutes les balles et est mise à jour avec la méthode GUI:: ballsUpdate().
- La scène supérieure qui correspond à la partie dynamique/interactive du jeu, c'est celle-ci qui va réagir aux entrées du joueur comme tous les mouvements, les boutons qui lance les mouvements introduits, ainsi que le boutons qui remet à 0 si les positions entrées ne sont finalement pas celles que le joueur voudrait envoyer.

À la fin du jeu, nous avons pensé à afficher un popup qui signale qui a gagné la partie. Après la fermeture de celui-ci, le jeu se ferme tout seul.

En conclusion, ce projet est un fruit de notre imagination pour lequel nous avons énormément travaillé mais il a également été un plaisir à réaliser pour tout ce que l’on a appris.