Bauwens Alexandre (39293)
Dupont Jules (40963)

Rapport de développement C++

Introduction :

Présentation du développement de l'application du jeu Quoridor en mode console par Jules Dupont et Alexandre Bauwens. Créé d'après l'énoncé du cours de C++ donné au quadrimestre 4 de l'ESI. Les règles du jeu et quelques adaptations pédagogiques ont dû être implémenté dans cette application. Le travail a nécessité une charge de travail d'environ 100h afin de créer le modèle, la vue, le concepteur, ainsi que l'ajout du design pattern Observateur/Observer, les lectures claviers fiables fournies par les codes de Nicolas Vansteenkist, l'écriture de la documentation et enfin les Tests fonctionnels.

Le programme a été développé avec l'IDE Qt Creator et avec un répertoire de stockage et développement Git. La version est principalement destinée à usage sur les plateformes Windows. Il ne serait pas impossible d'envisager un build pour les systèmes Apple et Linux.

Les classes modèles :

- Frame : Cette classe est utilisée afin de faire la représentation des cases du plateau de jeu. Dans notre version de jeu, il s'agit d'une classe abstraite ayant deux classes filles (PlayerFrame et WallFrame). Une case à la possibilité d'être remplie ou vide caractérisé par un booléen, elle a aussi une direction qui définit sur quelle place du contour du tableau elle se trouve.

  En règle générale, une WallFrame est toujours de direction Blank et seules les PlayerFrame peuvent avoir une valeur de la rosace des vents sur le contour (N S W E) dont les coins (NO NE SO SE). Les PlayerFrame qui ne sont pas sur le contour sont à Blank. Une méthode **isFree**() peut vérifier si une case est occupée.
- Board : Cette classe sert à représenter le plateau de jeu. Il s'agit d'un tableau en 2D de pointeur de Frame. Il a une taille en case joueur et une taille cachée ajoutant les cases accueillant les murs entre les cases de pions. Un plateau ne peut avoir de taille qu'entre 5 et 19 compris et ne peuvent qu'avoir une taille impair.

  La classe Board peut utiliser des méthodes de Frame de manière contrôlé afin de faire des actions sur les cases. On peut ainsi placer des pions ou des murs ou les retirer, vérifier si elles sont vide. Une méthode **toString**() permet de représenter le plateau et son contenu en mode console. Board possède une méthode particulière lui permettant de rechercher à partir d'une

position et d'une direction s'il existe au moins un chemin vers la direction donnée.

Pour être développé, l'algorithme de ***Pledge*** a été utilisé. Le principe est centré sur la recherche d'un chemin dans un labyrinthe à l'aveugle. On utilise un compteur et on vérifie ainsi ce qui se trouve face à soi et à notre gauche. Quand le compteur est à 0 on ne fait qu'avancer jusqu'à un obstacle. Si on s'arrête à cause d'un obstacle face à soi, on se tourne vers la droite et le compteur s'incrémente. S'il arrivait que ce compteur atteigne 15, on admet être bloqué et aucun chemin n'existe. Quand notre compteur est différent de 0, si on arrive au bout d'un obstacle sans se cogner en avancant et étant donc avec une ouverture sur notre gauche, on se tourne vers la gauche et dans ce cas-ci on décrémente le compteur (cela afin d'éviter de rester coincer sur un îlot). Après chaque déplacement de case on vérifie si on est arrivé à notre destination sinon on évalue son bras gauche et puis son nez.

- Side : Il s'agit d'une classe d'énumération permettant de définir des directions décrites par la rose des vents : North South West East NorthWest NorthEeast SoutWest SouthEast et Blank pour rien. Une méthode toString lui est ajoutée afin de pouvoir faire une représentation de direction dans la console.

- Player : La classe Player définit les moyens de représenter un joueur dans une partie de jeu Quoridor. On lui donne un nom, un numéro, une position, un stock de mur et une direction d'objectif. Un booléen sert aussi à décrire s'il est le gagnant de la partie ou non. Le constructeur a basiquement besoin du nom du joueur pour le qualifier mais afin de lui donner les bonnes quantités de mur et la bonne position de départ, on lui fournit, la taille du plateau de jeu, son numéro d'ordre et le nombre de joueur dans la partie. Cette classe possède donc en gros principalement des accesseurs et mutateurs.

- Game : Il s'agit de la classe centrale qui définit ce qu'est le jeu. Il y a un plateau de jeu et une liste de joueur. Une variable sert également à savoir dire quel est le numéro du joueur qui joue son tour. Deux constructeurs sont implémenter ne différent que par le nombre de string pouvant être passé en paramètre. Soit deux string pour une partie à 2 joueurs soit 4 pour 4 joueurs. Le dernier paramètre donne la taille que l'on désire donner à la construction du tableau.

  Des accesseurs sont bien sûr définit sinon plusieurs méthode particulières sont mises en place. **IsOver**() vérifie qu'une partie est terminée ; **collisionWall**() et **collisionPiece**() qui vérifie qu'une case n'est pas occupée dans la direction donnée ; **possiblePosition**() qui évalue toutes les positions vers lesquelles peut se déplacer légalement un pion en lui envoyant un set de

toutes ces directions ; **victoryCond**() vérifie si un joueur a atteint son objectif ; **playWall**() permet le placement complet et sans problème d'un mur dans le plateau de jeu et renvoie un booléen certifiant qu'il a bien été placé. Pour se faire, on place directement le mur (si collision de placement erreur) on évalue ensuite avec l'algorithme de *Pledge* (**findPath** de Board) qu'on ne bloque pas un des joueurs vers son objectif sans quoi on retire alors le mur. Si ça s'est déroulé correctement, on passe au joueur suivant et bien sur le mur est décompté du joueur courant ; **move**() fait la recherche des positions possibles où le joueur peut aller prenant en compte les obliques et les saut de pions. Comme on ne bouge pas réellement d'objet joueur dans le tableau, on modifie les emplacements de positions du joueur et on déplace la représentation dans le tableau en vidant le booléen de l'ancienne position et en remplissant la nouvelle. Le joueur et son pion modifie leurs positions en parallèle ; Enfin il y a une méthode **stringBoard**() faisant appel au **toString** de Board pour afficher le plateau via la Game.

Les classes vues :

- QuoridorConsole : Il s'agit de la classe gérant la vue du jeu permettant à l'utilisateur d'interagir avec le jeu. Elle s'occupe de traiter les informations envoyée par l'utilisateur, de les envoyer au classes métier qui traitent l'information. Cette classe sert également à afficher les résultats renvoyés par les classes métiers.

Les classes contrôleurs :

- Subject : -D'après la doc de la classe subject-

  Classe de base de tout "sujet d'observation » Classe dont dérive toute source d'événement (ou "sujet d'observation") du modèle de conception "Observateur / Sujet D'Observation".

- Observer : -D'après la doc de la classe observer-

  Classe abstraite de base de tout observateur. Classe dont dérive tout écouteur (ou "observateur") du modèle de conception "Observateur / SujetDObservation".

Les Tests :

Tests sur joueurs

TestPlayerCase1 : Création simple d'un joueur partie à quatre joueurs.

TestPlayerCase2 : Création simple d'un joueur partie à deux joueurs dans un plateau 9x9.

TestPlayerCase3 : Création d'un joueur qui envoie une erreur.

TestPlayerWallStock : Multiple tests sur le stock de murs: enlever 1 puis tous les murs et envoie d'erreur quand stock vide.

TestPlayerPosition : Test sur l'attribution d'une position de départ.

TestPlayerConstrFail : Test erreur construction de joueur.

TestPlayerConstrFail2 : Autre test d'erreur à la construction de joueur.

TestPlayerConstrOk : Test de construction de joueur fonctionnel.

TestPickWall : Test d'enlèvement de mur du stock.

TestGetName : Test de récupération du nom d'un joueur.

TestGetNum : Test de récupération du numéro d'un joueur.

TestHasWonSetwin : Test de la victoire d'un joueur avant et après la mise en victoire de ce joueur.

Tests sur cases

TestFrameSide : Test création d'une case pion et récupération de son orientation sur les contours.

TestFrameSide2 : Test création d'une case pion dans l'intérieur du plateau et récupération de son orientation.

FrameFailConstrP : Test d'erreur de construction de case joueur avec position incorrecte.

TestFramePrintContent : Test initialisant une case avec un placement et retournant le string du contenu.

TestFramePPlace : Affichage avant/après placement d'un joueur pour vérification.

TestFrameWPlace : Affichage avant/après placement d'un mur pour vérification.

TestFrameWEmpty : Affichage avant/après suppression d'un mur pour vérification.

Tests du Plateau

TestBoardCreateOK : Test Création d'un plateau OK.

TestBoardCreateKO : Test Création d'un plateau KO.

TestBoardIsFree : Tests de position libres.

TestBoardIsFree2 : Test de position après placement de pion et mur.

TestPlaceBoard : Place un pion en (0, 0); un mur en (1, 5) horizontale et un autre en (1, 1) vertical.

TestPlaceBoardFail : Test d'erreur après placement dans un plateau.

TestBoardEmptyWall : Test de vidage de case mur.

TestBoardEmptyWallFail : Test avec erreur de vidage mur.

TestGetlen : Test de récupération de la taille du plateau.

Test findPath sur Spirale

TestFindPath : Test sur la recherche de chemin dans une spirale avec chemin disponible.

TestFindPathBlocked : Test sur la recherche de chemin dans une spirale avec résultat bloquant.

TestFindPathFig4 : Test recherche chemin ouvert selon la figure 4 de l'énoncé du projet.

Test l'évaluation des positions possibles

TestEvalPosition : Test évaluant les positions possibles.

TestIsOverWinGame : Test affichant le joueur gagnant quand il atteint son objectif de victoire.

Test sur la classe d'énumération Side

TestSideToString : Test d'affichage de chaîne sur un side avec toString.

Test sur game

TestGameConstr : Test de création d'un jeu sans problème avec affichage du nom pour vérifier.

TestGameConstrFail : Test de création d'un jeu avec une dimension erronée .

TestGameNbPlayer : Test donnant le nombre de joueur d'un jeu créé avec 4 joueurs.

TestGetCurrentPlayerAndName : Test donnant le joueur courant d'un jeu pour vérifier qu'il change de joueur après le move en donnant le nom.

TestGameMove : Test vérifiant que le move lance bien le joueur suivant.

TestVictoryCond : Test vérifiant que la condition de victoire est mise à vrai si un joueur est sur sa destination de victoire.

TestPlayWall : Test vérifiant que les murs sont placés sauf si il bloque complètement un joueur.

Conclusion :

La première partie du développement a permis la découverte et l'apprentissage du langage C++ de manière très ludique grâce au sujet étant l'implémentation d'un jeu de plateau multijoueur. L'entre-aide par binôme est également un apprentissage là où le milieu de l'entreprise requiert souvent la collaboration de multiples personnes. Il est clair que ce projet ne s'est pas développé qu'entre deux étudiants attribué en binôme. Les échanges, discussions et conseils avec les autres élèves ont bien eut lieu, ainsi que l'aide par les professeurs ou les aides extérieures. On peut conclure que cette première version d'implémentation du jeu Quoridor en mode console est fonctionnel respectant les consignes de mise en œuvre et les règles du jeu tel que décrite dans l'énoncé du projet. Il y a eu apports comme la proposition des directions qu'un joueur peut prendre ou des détournements comme l'emploie de l'algorithme de Pledge plutôt que le backtracking.