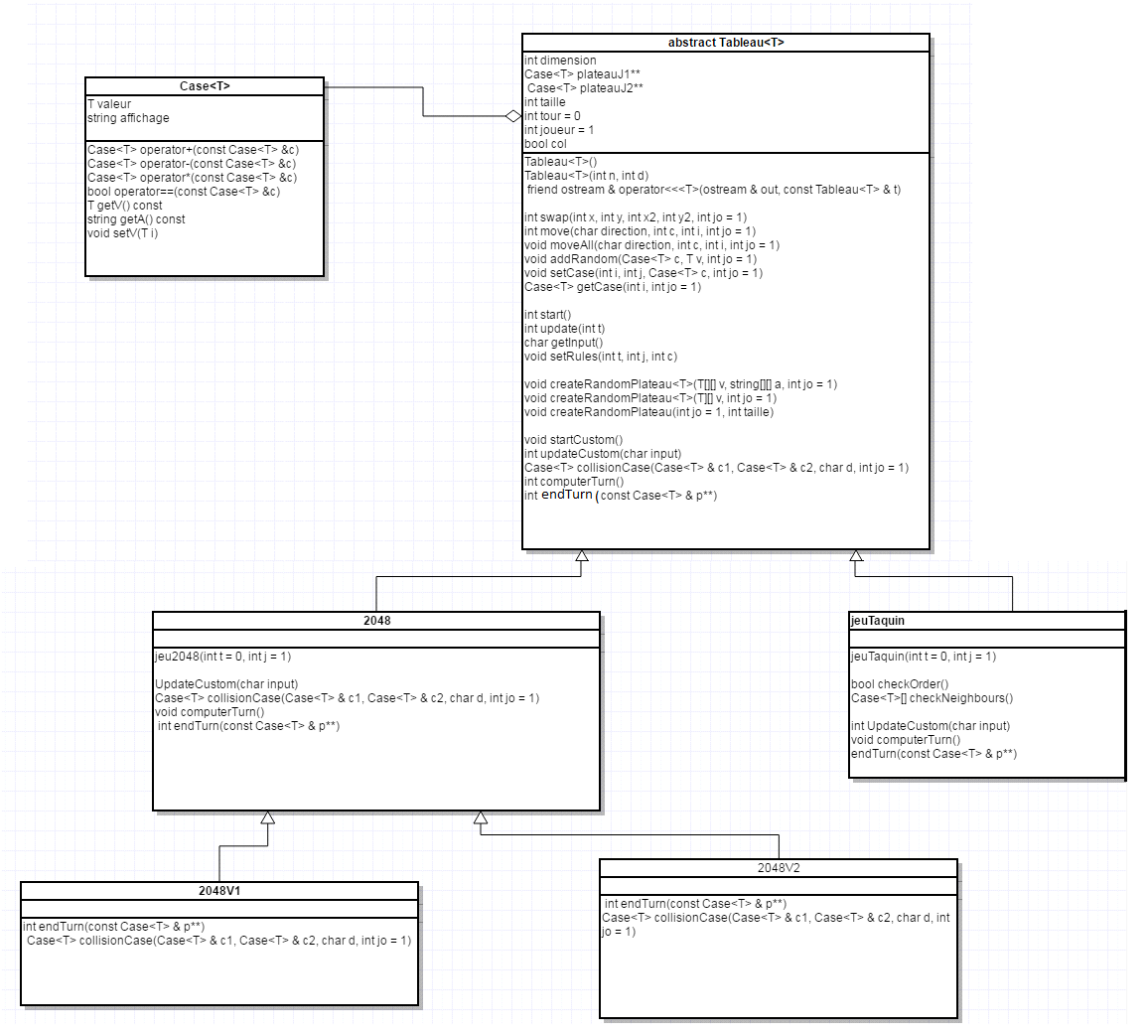
Le diagramme UML :



Les Fonctions et les variables sont présentées dans ce document dans le même ordre que dans le diagramme.

Présentation des Classes et leurs attributs et méthodes :

Il y a 2 classes de base (la classe Case et la classe Tableau), la classe Tableau sert d'usine pour construire les jeux puis on peut inclure les règles des jeux que l'on veut créer en surchargeant des fonctions prévues pour simuler des tours dans une partie et en utilisant les fonctions pour manipuler le plateau de jeu. Ensuite on montre comment utiliser les fonctions mises à notre disposition pour créer un jeu de Taquin ainsi que le 2048 et deux variantes du 2048.

Classe Case :

Variables :

string affichage : Ce qu'il y a d'afficher dans la case.

T valeur : La valeur de la case.

Fonctions :

Surcharge des opérateurs +, * et - : En fonction du type, ces surcharges effectuent différentes opérations mais elles retournent toutes un objet Case. Si le type est un nombre (int, float ou double), on additionne, multiplie ou soustraie les valeurs de 2 Case. Si le type est string ou char, les valeurs sont concaténées. Ensuite un objet Case est retourné avec la nouvelle valeur.

bool opérateur== (const Case<T> &c) : La fonction compare l'égalité des variables "valeur" de 2 Case et retourne un bool.

T getV() const et void setV(T v) : getter et setteur de la variable valeur.

T getA() const et void setA(T a) : getter et setteur de la variable affichage.

Classe abstraite Tableau :

La classe tableau prend en argument de son template <T>.

Variables :

int dimension : la dimension de chaque case des plateaux. La valeur maximal de cette variable est 4.

Case<T> plateauJ1** : Toutes les cases du plateau de jeu de taille n*n pour le joueur 1 (joueur humain).

Case<T> plateauJ1** : Toutes les cases du plateau de jeu de taille n*n pour le joueur 2 (contrôlé par l'ordinateur).

taille : La taille des plateaux.

int tour = 0 : Nombre de tour (0 pour un nombre illimité de tour)

int joueur = 1 : Nombre de joueur (max 2)

bool col : Autorise ou pas l'utilisation de la fonction CollisionCase.

I) Fonctions à ne pas surcharger :

1) Constructeurs et surcharge d'opérateurs :

Tableau<T>() : Constructeur par défaut (vide) de Tableau.

Tableau<T>(int n, int d) : n*n est la taille des plateaux et d la dimension de chaque Case.

friend ostream & operator<<<T>(ostream & out, const Tableau<T> & t) : Affiche le ou les plateaux sur le terminal en fonction des variables affichage des Cases, de la dimension des Cases et du nombre de joueurs (on affiche 2 terminaux s'il y a 2 joueurs). Par défaut, la dimension des cases correspond à la plus longue chaine de caractère parmi les variables d'affichage et la dimension est également borné à 4. Si la plus grande chaine de caractère à une taille supérieure à 4, alors seuls les 4 premiers caractères de la chaine seront représentés dans le plateau.

2) Fonction pour créer des algorithmes pour les jeux :

int swap(int x, int y, int x2, int y2, int jo = 1) : Intervertit deux Case du plateau ayant les coordonnées en argument. int jo permet de cibler le tableau à modifier, 1 pour le tableau du joueur 1 (joueur humain), 2 pour l'adversaire et 3 pour les deux, idem pour les fonctions suivantes.

int move(char direction, int c, int i, int j, int jo = 1) : Déplace une Case dans le plateau vers la direction souhaitée. char direction peut avoir les valeurs suivantes : 'd', 'g', 'h' et 'b', elles déterminent dans quelle direction se déplace la Case. int c est la distance que doit parcourir la Case. int i et int j sont les coordonnées de la Case. Inclue la fonction CollisionCase() que l'utilisateur doit remplir et qui est lancée à chaque fois que 2 Case rentrent en collision à condition que la variable fus soit égale à true. Retourne 0 si la Case n'a pas pu bouger, sinon 1.

void moveAll(char direction, int c, int i, int j, int jo = 1) : Similaire à la fonction move mais pour toutes les Case du plateau. Retourne 0 si aucune des Case n'a pu bouger, sinon 1.

void addRandom(Case<T> c, T v, int jo = 1) : Choisie au hasard une case ayant la valeur v du plateau et y ajoute c.

void setCase(int i, int j, Case<T> c, int jo = 1) et Case<T> getCase(int i, int j, int jo = 1) : getter et setter d'un emplacement du plateau.

3) Mise en route du jeu :

int start() : Permet au joueur de démarrer le jeu. Cette fonction lance la fonction Update(int t).

void update() : Il y a une boucle while qui continue indéfiniment tant que le jeu ou que le nombre de tours maximal n'est pas atteint. La boucle affiche le ou les plateaux en fonction du nombre de joueurs et

les met à jour à la fin de chaque tour. Au début de chaque boucle, la fonction UpdateCustom(char input) est appelée. A la fin de la boucle, la fonction computerTurn est lancée (voir II)) suivit de la fonction endTurn vérifiant les conditions de fin de partie (voir II)), si endTurn retourne 1 la partie continue, si elle retourne 0, la partie se termine par une victoire, si elle retourne -1 c'est une défaite. Ensuite le terminal est nettoyé à la fin de la boucle.

char getInput() : Récupère l'input du joueur. Cette fonction retourne les valeurs (si le joueur appuit sur les touches correspondantes) : 'd', 'z', 'q', 's' (pour les 4 directions), 'escape' (pour arrêter le jeu en cours).Si une touche non attendue est utilisée par l'utilisateur, la fonction retourne une exception et le programme s'arrête.

void setRules(int t, int j, bool c) : Etablis le nombre de tours t, le nombre de joueurs j et c représente l'utilisation ou non de la fonction CollisionCase (voir à la suite).

4) Création du plateau :

Ces fonctions vérifient si les tableaux en paramètre ont une taille crédible, sinon on retourne une exception.

void createRandomPlateau<T>(T[][] v, string[][] a, int jo = 1) : Créé un plateau avec des Cases ayant les valeurs v et les variables d'affichages a. Les Cases sont placées aléatoirement dans le tableau.

void createRandomPlateau<T>(T[][] v, int jo = 1) : Créé un plateau avec des Cases ayant les valeurs v et les variables d'affichages sont les variables valeurs converties en string. Les Cases sont placées aléatoirement dans le tableau.

void createRandomPlateau(int jo = 1, int taille) : Créer un plateau avec des nombres aléatoire (entre 1 et 100). Les variables d'affichages sont les variables valeurs converties en string. int jo permet de cibler le tableau à modifier, 1 pour le tableau du joueur 1 (joueur humain), 2 pour l'adversaire et 3 pour les deux.

II) Fonctions à surcharger par l'utilisateur :

Ces fonctions sont **vides** et doivent être surchargées par l'utilisateur afin qu'il puisse créer son jeu.

void startCustom() : Permet à l'utilisateur d'instancier son jeu.

void updateCustom(char input) : char input est une variable qui prend en paramètre ce qu'à été entré par l'utilisateur dans le terminal et vérifié par la fonction getInput(). Le programmeur devra surcharger cette fonction pour établir les règles de son jeu.

void collisionCase(Case<T> & c1, Case<T> & c2, char d) : Cette fonction est utilisée à condition que col = true. Elle est lancée lorsque une 2 Case rentrent en collision lors de l'utilisation de la fonction move ou moveAll. La variable char d correspond à la direction ('d', 'z', 'q', 's') dans lequel le mouvement s'est fait.

L'utilisateur peut indiquer dans cette fonction le comportement d'une collision. La Case c1 est celle qui bouge après c2 (donc c1 cogne dans l'arrière train de c2).

int computerTurn() : Ce qui se passe pendant le tour de l'ordinateur. La fonction est appelée dans update().

int endTurn(const Case<T> & p**) : L'utilisateur doit inscrire ici ce qui se passe à la fin d'un tour et les conditions de fin de partie. La variable p prend en paramètre le plateau du joueur humain ou celui de l'ordinateur. Cette fonction est appelée 2 fois dans la fonction update pour vérifier les conditions des parties des 2 joueurs :

Dans le cas du joueur humain :

La fonction retourne 0 si la partie est une victoire pour le joueur humain et -1 si c'est une défaite, **alors la partie s'arrête** (y compris celle de l'ordinateur) dans ces 2 cas. Sinon on retourne 1.

Dans le cas de l'ordinateur :

La fonction retourne 0 si la partie est une victoire et -1 si c'est une défaite, **la partie de l'ordinateur s'arrête mais pas celle du joueur humain** (on ne passe plus par la fonction computerTurn())dans ces 2 cas et un message dans le terminal indique que l'ordinateur a fini sa partie. Sinon on retourne 1.

Classe jeuTaquin :

La classe jeuTaquin hérite de Tableau.

1) Constructeur

jeuTaquin(int t = 0, int j = 1) : Initialise le jeu, fait appel à la fonction setRules(t, j, false), puis créé le tableau avec createRandomPlateau(). Ensuite on détermine ou est l'espace vide du plateau en prenant une case au hasard puis on met une valeur de 0 à cette Case et un affichage vide (un string "").

2) Mise en marche du jeu

bool checkOrder(const Case<T> & p**) : Vérifie si les cases du plateau sont dans l'ordre

Case<T>[] checkNeighbours(const Case<T> & p**) : Retourne les voisins de la case vide

int updateCustom(char input) (Surcharge) : Une direction est indiqué via la fonction Update, la direction doit avoir pour valeur ces caractères : 'd', 'z', 'q' ou 's'. En fonction de la direction choisie, on déplace un voisin de la case vide vers la case vide grâce à la fonction swap et checkNeighbour. Puis on vérifie si les cases sont dans l'ordre grâce à checkOrder, si oui on retourne 0 sinon 1.

computerTurn() (Surcharge) : Sélectionne un voisin de la case vide au hasard (sur son plateau) et le swap par la case vide.

endTurn(const Case<T> & p**) (Surcharge) : checkOrder pour le plateau en paramètre et retourne 0 en

cas de victoire, sinon retourne 1.

Ensuite on peut démarrer le jeu dans une fonction main grâce à la fonction start().

Classe jeu2048 :

La classe jeu2048 hérite de Tableau.

1) Constructeur

jeu2048(int t = 0, int j = 1) : Initialise le jeu, fait appel à la fonction setRules(t, j, true), puis créé le (ou les) tableau avec createRandomPlateau() en passant en paramètre un tableau d'entier de 0, mais pas de tableau de string donc l'affichage correspond aux valeurs.

2) Mise en route du jeu

int updateCustom(char input) (Surcharge) : On vérifie la direction qui est indiquée par l'input, puis on appelle la fonction moveAll avec input en paramètre.

void collisionCase(Case<T> & c1, Case<T> & c2, char d, int jo = 1) (Surcharge) : On multiplie les Case entre elles et on remplace par une Case ayant une valeur de 0 la Case c1. On rappelle la fonction moveAll afin que toutes les Case se déplacent bien y compris dans les Case vides générées par les multiplications de Case.

int computerTurn() (Surcharge) : On utilise la fonction moveAll avec une direction aléatoire. Si moveAll retourne 0, on réessaye de sélectionner aléatoirement une direction (sans la ou les directions(s) choisit auparavant).

int endTurn(const Case<T> & p**) (Surcharge) : Au début de la fonction, on appelle la fonction addRandom pour ajouter une case de valeur 2 ou 4 (sélectionner aléatoirement, en indiquant leur variable d'affichage).

Retourne -1 si aucun mouvement n'est possible, sinon 0 si une case a pour valeur 2048, sinon 1 pour continuer la partie.

Classe jeu2048V1 :

La classe jeu2048V1 hérite de jeu2048. Cette classe ajoute une nouvelle règle au 2048, on peut voir apparaitre des Case ayant pour valeur -1 (et étant affiché aussi -1) qui permet de supprimer une Case lorsque la Case -1 rentre en collision avec une autre Case. il faut donc mettre à jour la fonction endTurn

et CollisionCase. Cette nouvelle Case a pour variable d'affichage "Dest" pour Destroy.

int endTurn(const Case<T> & p**) (Surcharge) : Au début de la fonction, on appelle la fonction addRandom pour ajouter une case de valeur 2, 4 **ou -1** (sélectionner aléatoirement, en indiquant leur variable d'affichage).

Retourne -1 si aucun mouvement n'est possible, sinon 0 si une case a pour valeur 2048, sinon 1 pour continuer la partie.

Case<T> collisionCase(Case<T> & c1, Case<T> & c2, char d, int jo = 1) (Surcharge) : **Si l'une des Case a pour valeur -1, on supprime les 2 Case et on retourne une Case ayant pour valeur 0. Sinon** on multiplie les Case entre elles et on remplace par une Case ayant une valeur de 0 la Case opposée à la direction. On rapelle la fonction moveAll afin que toutes les Case se déplacent bien y compris dans les Case vides générées par les multiplications de Case.

Classe jeu2048V2 :

La classe jeu2048V2 hérite de jeu2048. Cette classe ajoute une nouvelle règle au 2048, on peut voir apparaitre des Case ayant pour valeur 20 ou -20 et pour variable d'affichage *2 ou /2. On appellera la Case *2 celle ayant pour valeur 20 et la Case /2 avec la valeur -20. La Case *2 multiplie par 2 la Case avec laquelle elle rentre en collision et la Case /2 divise par 2. On doit pouvoir faire apparaitre ces nouvelles Case et appliquer leur propriété, donc comme pour jeu2048V1, on surcharge updateCustom et collisionCase.

int endTurn(const Case<T> & p**) (Surcharge) : Au début de la fonction, on appelle la fonction addRandom pour ajouter une case de valeur 2, 4, **20 ou -20** (sélectionner aléatoirement, en indiquant leur variable d'affichage).

Retourne -1 si aucun mouvement n'est possible, sinon 0 si une case a pour valeur 2048, sinon 1 pour continuer la partie.

Case<T> collisionCase(Case<T> & c1, Case<T> & c2, char d, int jo = 1) (Surcharge) : **Si l'une des Case a pour valeur 20 ou -20, alors on multiplie ou on divise la case adjacente sauf si l'autre Case a également pour valeur 20 ou -20, dans ce cas on ne fait rien.**

Sinon on multiplie les Case entre elles et on remplace par une Case ayant une valeur de 0 la Case opposée à la direction. On rapelle la fonction moveAll afin que toutes les Case se déplacent bien y compris dans les Case vides générées par les multiplications de Case.