Mini-Projet : jeu 2048 (à faire en binôme)

Le projet comprend une base à effectuer par tous les groupes ainsi que des extensions qui ne sont à débuter qu'une fois que le cœur du système aura été réalisé. Il ne sert à rien de développer des extensions tant que le cœur du système n'a pas été terminé ou testé correctement. Les extensions ne rapporteront aucun point dans ce cas de figure.

Le jeu 2048 est une variante du *jeu de taquin*, inventée par le web designer Gabriele Cirulli en 2014. Le jeu original est une grille 4×4 où chaque case est soit vide, soit contient une tuile étiquetée par une puissance de 2. Le but du jeu est de faire glisser (verticalement ou horizontalement) toutes les tuiles (en même temps) sur la grille afin de combiner les tuiles de mêmes valeurs. Si deux tuiles de mêmes valeurs (2^k) sont adjacentes pendant le glissement, alors elles se combinent en une unique tuile étiquetée par la somme des valeurs (2^{k+1}). Après chaque déplacement, une nouvelle tuile apparaît aléatoirement sur un des emplacements vides. Cette nouvelle tuile a pour valeur soit 2, soit 4, avec probabilité respective $\frac{9}{10}$ et $\frac{1}{10}$. Le jeu commence avec deux tuiles placées aléatoirement sur la grille ; il se termine soit lorsque le total est atteint (2048 dans la version initiale), soit lorsque toutes les cases sont occupées et qu'aucun mouvement ne permet de combiner de tuiles.

Objectif : Rendre un programme mettant en oeuvre les bases du jeu 2048, en introduisant quelques variantes par rapport à la version originale [1].

Bases du jeu

L'action de base du jeu consiste donc à provoquer des glissements des tuiles actuelles de façon à fusionner deux tuiles "adjacentes" (elles peuvent être séparées par des cases vides) de mêmes valeurs. A chaque action on provoque le glissement en indiquant par une touche la direction de ce glissement. Qu'on ait pu procéder ou non à des fusions, l'étape se termine par un compactage de la grille dans la direction indiquée : les tuiles sont déplacés de façon à ne pas laisser de case vide entre deux tuiles dans cette direction.

Le principe du jeu est clair et le jeu est accessible sur Internet. Nous apportons quelques précisions pour lever certaines ambiguïtés. Nous illustrons les situations avec un déplacement vers la droite sur une seule ligne. Les autres cas sont similaires :

- A chaque étape, une tuile ne peut faire l'objet que d'une fusion : une tuile qui résulte de la fusion de deux tuiles ne peut pas, dans la même étape, fusionner avec une autre tuile ;
- A chaque étape il est possible de faire plusieurs fusions sur la même ligne ou colonne tant que ces fusions s'appliquent à des tuiles différentes
  L'exemple ci-dessous illustre les deux règle précédentes, avec deux fusions indépendantes :

```
*************************                    *************************
*    4 *   4 *   4 *   4 *     donne         *     *       *   8 *   8 *
*************************                    *************************
```

- Quand on a le choix de fusionner une tuile avec l'une ou l'autre de ses deux tuiles adjacentes, on commence les fusions par les tuiles à l'extrémité de la ligne/colonne dans la direction du mouvement. Dans l'exemple ci-après, puisqu'il s'agit d'un déplacement vers la droite, la tuile du milieu fusionne avec celle à sa droite :

1. L'aspect plaisant du jeu est lié à la taille de la grille, la valeur à atteindre et les proportions respectives des tuiles 2 et 4, de façon à rendre le jeu ni trop simple, ni trop long. Nous n'avons pas de telles considérations ici.

**********		**********
* 4 * 4 * 4 * *	donne	* * * 4 * 8 *
**********		**********

De même dans le premier exemple, ce ne sont pas les deux tuiles du milieu qui ont fusionné, sinon la ligne contiendrait deux tuiles 4 qui encadreraient une tuile 8.

Réalisation

Le découpage du programme en fonctions est imposé mais vous pouvez ajouter des fonctions auxiliaires. Votre réalisation doit respecter les en-têtes des fonctions, ainsi que les comportements indiqués. Votre réalisation era soumise à un programme de test basé sur ces fonctions. Si vous changez le comportement de ces fonctions, votre programme échouera aux tests.

Ces fonctions permettent de réaliser l'application sans se préoccuper du « menu interactif » : vous pouvez vous en servir pour construire des grilles test à passer en paramètre à vos fonctions. Vous pouvez ainsi tester des situations qu'il serait problématique d'atteindre en se basant uniquement sur les actions de jeu. Les commandes utilisateur et l'aspect aléatoire permettent dans un second temps d'enrichir votre démonstration avec des situations non anticipées et de réaliser un vrai jeu. On ne se focalisera donc pas dans un premier temps sur la réalisation d'une interface utilisateur.

Une instance du jeu est décrite par le type `Grille` que vous devez définir. Des fonctions imposées permettent d'accèder aux informations intéressantes sur la configuration du jeu. Toutes les fonctions liées au jeu prennent en paramètre un référence sur une grille.

Fonctions à implémenter

- `void init(Grille &g, size_t dimension, size_t cible, size_t proportion)` : initialise `g` avec les paramètres indiqués ; la grille résultat doit comporter les deux premières tuiles dont les valeurs et positions sont obtenues par des appels aux fonctions `nouvelle` et `place` décrites ci-après ;
- `bool charge(Grille &g, const vector<vector<size_t>> &v)` : initialise une grille à partir du vecteur de ses lignes, chacune étant décrite par le vecteur des valeurs de ses tuiles, de gauche à droite (une valeur 0 désigne une case vide). La fonction n'est applicable que si le score est vierge et les vecteurs de dimensions compatibles avec la dimension de la grille. La fonction renvoie *true* si la fonction est applicable et *false* sinon, auquel cas la grille de départ est inchangée.
- les fonctions `int gauche(Grille &g)`, `int droite(Grille &g)`, `int haut(Grille &g)`, `int bas(Grille &g)` pour les quatre actions de glissement. Les fonctions procèdent aux fusions de tuiles dans la direction indiquée, au compactage et renvoient le nombre de cases vides dans la grille. Si le mouvement dans la direction demandée est impossible, la fonction renvoie -1 et laisse la grille inchangée (on n'ajoute pas de nouvelle tuile).
- `void affiche(const Grille &g)` : provoque l'affichage de la grille.

Les fonctions ci-dessous permettent de connaître l'état du jeu sans dépendre de votre implémentation du type `Grille` :

- `bool succes(const Grille &g)` : renvoie *true* ssi la valeur cible a été atteinte ;
- `size_t vides(const Grille &g)` : renvoie le nombre de case vides ;
- `size_t cible(const Grille &g)` : renvoie la valeur à atteindre ;
- `size_t score(const Grille &g)` : renvoie le nombre actuel de points ;
- `size_t proportion(const Grille &g)` : renvoie la proportion de tuiles 2 au tirage, exprimée par un entier entre 0 et 10 (/ 10) ;
- `size_t dimension(const Grille &g)` : renvoie la dimension de la grille.

Fonctions mises à disposition

- `size_t nouvelle(const Grille &g)` : renvoie la valeur de la prochaine tuile en respectant la proportion moyenne demandée de tuiles de valeur 2 ;

— `size_t place(const Grille &g)` : spécifie la case qui doit recevoir la prochaine tuile (la grille `g` doit avoir au moins une case vide). On considère que les coordonnées des cases vides de `g` sont implicitement stockées dans un vecteur obtenu lors d'un parcours de haut en bas et de gauche à droite de la grille. La fonction renvoie l'indice dans ce vecteur de la case à remplir par la prochaine tuile ajoutée ; la position est calculée à l'aide d'un "générateur de nombres pseudo-aléatoire"[2] en respectant une distribution uniforme sur l'ensemble des cases vides. Son fonctionnement est illustré comme suit :

```
************************
* 128 *  4  *     *     *
************************
*  2  * 16  *  4  *  2  *
************************
*     *  8  *  8  *  4  *
************************
*  2  *     *  2  *  4  *
************************
```

Ici, le vecteur des coordonnées des cases vides est `{ (1, 3), (1, 4), (3, 1), (4, 2) }`, `place(g)` va renvoyer un indice compris entre 0 et 3. Si le résultat ets 0, la prochaine tuile devra être placée sur la case de coordonnée `(1, 3)` de `g`. Si le résultat est 1, la prochaine tuile devra être placée sur la case `(1, 4)` de `g`, etc.

— `void resetRand(bool mode)` : Si `mode` vaut *true* le générateur de nombres aléatoires est initialisé avec une valeur aléatoire et fournit à chaque exécution une séquence différente de valeurs. Sinon, il est initialisé de façon à fournir toujours la même séquence de valeurs si on l'appelle dans les mêmes conditions. Cette fonction est à appeler **une fois et une seule au début de la simulation**. Utiliser la même séquence de nombres aléatoires facilite dans un premier temps la mise au point de votre programme en assurant la reproductibilité du comportement d'une exécution à l'autre. En mode "jeu", on ne veut bien sûr pas récupérer la même séquence à chaque partie et on appellera `void resetRand` avec `false`.

Interaction avec l'utilisateur

Dans la version console, on rentre chaque déplacement (*haut, bas, gauche* et *droite*) par l'intermédiaire de caractères (par exemple `h`, `b`, `g` et `d` ou encore `i`, `k`, `j` et `l`) suivi de `Entrée`. Ne pas se servir des touches de type "flèche" du clavier qui sont interprétées de façon spéciale.

On affiche la grille après chaque étape, une fois la nouvelle tuile insérée. On veillera à ce que les colonnes soient bien alignées. On supposera que la valeur d'une tuile n'occupe jamais plus de 5 symboles. Vous pouvez vous servir de l'instruction `printf("%5d", maValeur) ;` qui permet d'écrire la valeur indiquée cadrée à droite sur 5 positions, en complètant à gauche par des espaces (voir le second polycopié pour un exemple d'utilisation de `printf` et le fichier d'en-tête à inclure). Vous pouvez aussi utiliser la méthode `setw` de `stream` (en incluant le fichier d'en-tête `<iomanip>`) qui spécifie la largeur d'impression avec lequel écrire le prochain élément. Exemple : `cout << setw(5) << 13` qui écrira la valeur 13 cadrée à droite sur 5 caractères.

Voici un exemple possible d'exécution :

```
************************
* 128 *  4  *     *     *
************************
*  2  * 16  *  4  *  2  *
************************
*     *     *     *     *
************************
*  2  *     *     *     *
************************
```

2. voir une explication sur Wikipédia par exemple.

```
Entrer commande : h
************************
* 128 *  4  *  4  *  2  *
************************
*  4  * 16  *     *     *      fusion des cases (2, 1) et (4, 1)
************************
*     *     *     *     *
************************
*     *  2  *     *     *      apparition d'une tuile 2 en position (4, 2)
************************

Entrer commande : g
************************
* 128 *  8  *  2  *     *      fusion des cases (1, 2) et (1, 3) et
************************
*  4  * 16  *     *     *      glissement de la tuile (1, 4)
************************
*     *     *  4  *     *      apparition d'une tuile 4 en position (3, 3)
************************
*  2  *     *     *     *      glissement vers la gauche de la tuile (4, 2)
************************

Entrer commande : ...
```

Extensions

- réaliser un menu interactif en mode console, permettant de changer interactivement les valeurs des paramètres de départ, de lancer un test spécifique, de redémarrer une partie, etc. Un exemple de menu interactif est présent dans le premier polycopié comme illustration de l'instruction `switch` ;
- Sauvegarder la configuration courante dans un fichier textuel dont on passe le nom en paramètre, ainsi que restaurer une configuration à partir d'un fichier : le jeu reprend dans l'état dans lequel il était au moment de la sauvegarde.
- Ajouter un mode graphique un intégrant une librairie d'interface graphique sous CodeBlocks. Ceci nécessite la présence de certaines librairies et l'ajouter de paramètres de compilation.

Pour les deux dernières extensions, des indications seront données ultérieurement.

Organisation des fichiers

Afin de faciliter les tests et extensions ultérieurs, votre programme sera constitué d'au moins 4 fichiers : `jeu.h` contiendra toutes les définitions de type ainsi que les en-têtes des principales fonctions, `jeu.cpp` contiendra les fonctions demandées ainsi que vos éventuelles fonctions auxiliaires, `dispo.cpp` contiendra toutes les fonctions fournies par l'équipe enseignante (vous ne devez pas modifier ce fichier car il pourra évoluer si nous mettons de nouvelles fonctions à disposition), `test.cpp` contiendra vos fonctions de test, la fonction `main` ainsi que, ultérieurement, le menu interactif ou les extensions graphiques ;

Modalités d'évaluation

L'évaluation tiendra compte de la qualité de votre réalisation, d'un court document descriptif et d'une brève démonstration. Les documents à fournir seront précisés ultérieurement.

Le projet est à faire en groupe mais la note pourra être individualisée au vu de la participation de chacun à la réalisation et à la démonstration (tous les membres des groupes devront être présents et devront pouvoir répondre aux questions). **L'équipe pédagogique pourra utiliser des logiciels de calcul de similarités entre programmes pour mettre en évidence d'éventuels plagiats.**