Introduction à la Programmation

—

Examen partiel sur machine

—

G1: pascal.monasse(at)enpc.fr
G2: abdelkader.hamadi(at)enpc.fr
G3: thomas.belos(at)enpc.fr
G4: laurent.bulteau(at)u-pem.fr
G5: abderahmane.bedouhene(at)enpc.fr
G6: pierre-alain.langlois*(at)*enpc.fr
G7: marcela.carvalho(at)onera.fr

—

25/10/19

—

1 Enoncé

1.1 Le jeu 2048

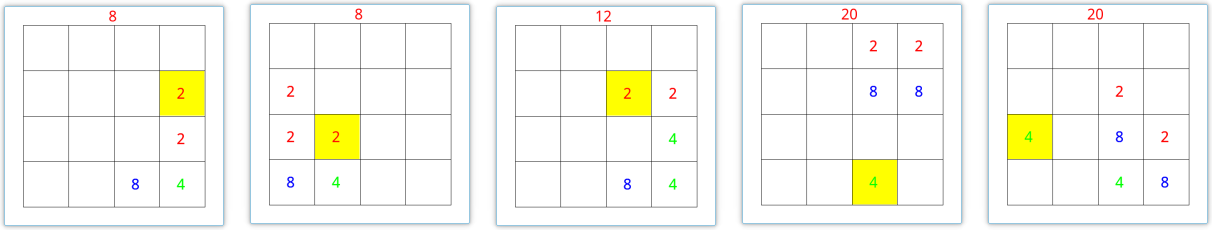
Nous allons programmer le jeu 2048, dont le principe consiste à combiner des nombres égaux adjacents dans une grille pour les remplacer par leur somme, soit le double. On part d'une grille $n \times n$ vide (normalement $n = 4$) et le nombre 2 ou 4 apparaît aléatoirement dans une case libre. Le joueur peut décider avec les flèches du clavier de faire glisser les lignes horizontalement ou les colonnes verticalement. Les cases occupées se collent les unes contre les autres et deux cases adjacentes de même valeur sont remplacées par une seule case avec leur valeur double, et le score est augmenté de ce double. Par exemple la ligne $(4 \quad 4 \quad 8 \quad 2)$ devient $(8 \quad 8 \quad 2 \quad .)$ par la flèche gauche et $(. \quad 8 \quad 8 \quad 2)$ par la flèche droite, le point désignant une case vide. Notez que les combinaisons ne sont pas cumulatives, les deux 4 devenant un 8 ne sont pas directement combinés au 8 adjacent. De même, $(4 \quad 4 \quad 4 \quad 4)$ devient $(8 \quad 8 \quad . \quad .)$. Le jeu s'arrête lorsqu'il n'y a plus de case libre au moment d'ajouter un nombre, et le but est d'obtenir le plus haut score. Faire apparaître le nombre 2048 est aussi un objectif, mais rien n'interdit d'aller au-delà. [1] Créez un projet Imagine++ avec un fichier contenant le `main`. Prenez le `CMakeLists.txt` d'un projet existant et adaptez-le.

Il est plus important de livrer un code clair (commenté et indenté) et qui compile sans warning, même s'il ne répond pas à toutes les questions. Pour cela, vérifiez à chaque étape que votre programme compile et se lance correctement. À la fin de l'examen, nettoyez votre code (indentation...) et vérifiez qu'il compile. Créez alors une archive portant votre nom et numéro de groupe.

1.2 Bases du jeu

1. La fenêtre réserve `marge`= 40 pixels de chaque côté de la grille, et chacune des $n \times n$ cases ($n = 4$) prend $z = 100$ pixels dans chaque dimension. Votre fonction `main` ouvre une fenêtre de la taille adéquate et appelle `endGraphics` en fin de programme. Ces valeurs sont définies comme des constantes.
2. Définir une fonction `dessine_cases` qui trace les cases (le quadrillage).
3. Dans un fichier à part, définir une structure `Grille` contenant le score actuel et le tableau des valeurs des cases.
4. Définir une fonction `init` qui initialise les champs de son argument de type `Grille`. Les cases vides sont codées par une valeur 0.

1. Le joueur arrivant à 2048 aurait pourtant mieux fait d'occuper son temps à des activités plus productives, tandis que le joueur voyant dans sa grille un nombre qui n'est pas puissance de 2 peut blâmer le programmeur pour un bug scandaleux.



Grille courante | Touche gauche ← | Touche droite → | Touche haut ↑ | Touche bas ↓

FIGURE 1 – À partir de la figure de gauche, les différentes flèches du clavier produisent successivement les résultats présentés. La case jaune qui apparaît dans les résultats est un nouveau nombre pour l'action suivante. Le nombre en-dehors de la grille est le score, qui augmente quand des additions ont lieu.

5. Définir une fonction `dessine` prenant une grille et affichant les valeurs des cases (mais pas les 0 des cases vides). Pour convertir un entier i en std :: string, utiliser `std::to_string(i)` (#include <string>). Utiliser une constante `txt`= 24 pour la taille des chiffres, et essayer de centrer l'affichage dans la case. La position passée à `Imagine::drawString` est le coin bas-gauche du texte.
6. Définir une fonction `libres` retournant le nombre de cases vides de la grille argument.
7. Définir une fonction `ajoute` qui choisit une case vide au hasard uniformément pour y mettre aléatoirement un 2 ou un 4. Elle retourne `false` s'il n'y a pas de case vide.
8. Tester le remplissage de la grille par ajouts successifs, chaque ajout attendant un clic de l'utilisateur.

1.3 Actions du joueur

Pour ne pas avoir à copier/coller des fonctions très similaires pour les quatre flèches, on va se concentrer sur la flèche gauche. Les autres se ramèneront à ce cas par pré- et post-traitement.

9. Définir une fonction `zeros_a_droite` qui place tous les 0 des lignes de sa grille argument sur la droite, les autres nombres restant dans leur ordre initial : la ligne $\begin{pmatrix}0 & 2 & 0 & 8\end{pmatrix}$ devient $\begin{pmatrix}2 & 8 & 0 & 0\end{pmatrix}$.
10. Définir la fonction `decale_gauche` qui gère la flèche gauche. Le principe est de commencer par appeler `zeros_a_droite`, puis parcourir la ligne de gauche à droite : lorsqu'un nombre est égal à son suivant, on double le premier et on met le deuxième à 0, puis on continue le parcours en sautant ce dernier. À la fin, on réapplique `zeros_a_droite`.
11. Définir une fonction `miroir` qui inverse chaque ligne : $\begin{pmatrix}0 & 2 & 0 & 8\end{pmatrix}$ devient $\begin{pmatrix}8 & 0 & 2 & 0\end{pmatrix}$.
12. Définir la fonction `transpose` qui transforme les colonnes de la grille en lignes, comme la transposée d'une matrice : $G'_{ij} = G_{ji}$.
13. Définir la fonction `decale` qui prend une grille et un code de touche (une touche flèche du clavier) et appelle miroir et/ou transpose si approprié, utilise `decale_gauche` et refait les opérations inverses. [2]
14. Le jeu doit être jouable, programmer donc le `main`. Utiliser `Imagine::getKey` pour attendre l'appui d'une touche au clavier. Par exemple, la flèche gauche renvoie la constante `Imagine::KEY_LEFT`.

1.4 Améliorations

15. Quand un nouveau nombre apparaît dans `ajoute`, on veut qu'il s'affiche sur fond jaune. Ajouter le repère de la case dans `Grille` et le gérer dans `init`, `dessine`, `ajoute` et `decale`, cette dernière fonction mettant des coordonnées négatives car on n'a pas encore rajouté un nouveau nombre.
16. Gérer le score et son affichage dans la marge supérieure de la fenêtre, affichage à peu près centré.
17. Dans le `main`, prévoir une fonctionnalité *undo* : avant chaque action du joueur, on sauve la grille dans une variable auxiliaire. Lors du `getKey` suivant, la touche de correction *backspace* (code `Imagine::KEY_BACK`) remet la grille précédente.
18. Synesthésie : attribuer des couleurs aux nombres. Prévoir un tableau de 7 couleurs (rouge, vert, bleu, etc). Écrire une fonction `couleur` qui renvoie la première couleur pour le nombre 2, la deuxième pour le nombre 4, etc. Au-delà du nombre $2^7 = 128$, boucler dans la liste. Utiliser cette fonction dans `dessine`.

2. Notez que le miroir et la tranposition sont involutives, leur propre inverse.