Introduction à la Programmation

—

Examen final sur machine

—

G1: monasse*(at)*imagine.enpc.fr G2: nicolas.audebert*(at)*onera.fr
G3: alexandre.boulch*(at)*onera.fr G4: maxime.ferrera*(at)*onera.fr
G5: laurent.bulteau*(at)*u-pem.fr G6: pierre-alain.langlois*(at)*eleves.enpc.fr

—

12/01/18

—

1 Enoncé

1.1 Automate cellulaire

Les automates cellulaires, imaginés par John von Neumann (1903-1957) et développés par John Conway (1937-), simulent suivant des règles simples l'évolution de populations. On a une grille de $w \times h$ cellules, chacune pouvant être soit vivante (représentée par un carré noir) soit morte (laissée en blanc). On part d'une situation initiale avec quelques cases vivantes, et on observe l'évolution, qui obéit aux règles suivantes :

1. Une cellule vivante avec pas plus d'un voisin vivant meurt d'isolement.
2. Une cellule vivante avec au moins 4 voisins vivants meurt de surpopoulation.
3. Une cellule morte avec *exactement* 3 voisins vivants devient vivante (reproduction).
4. Dans tout autre cas, la cellule reste dans le même état.

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.

Créez un projet Imagine++ avec un fichier contenant le `main`. Prenez le `CMakeLists.txt` d'un projet existant et adaptez-le.

1.2 Classe de base

1. Dans des fichiers à part, créer une structure `Cellule`, gérant pour l'instant juste son état (variable booléenne).
2. Créer une classe `Grille` contenant un tableau de $w \times h$ cellules (w et h sont des variables).
3. Lui ajouter un constructeur prenant en argument les valeurs w et h (allocation dynamique de mémoire).
4. Lui ajouter le destructeur.
5. Ajouter une méthode `set` permettant de fixer l'état d'une cellule (x, y) à vivant ou mort.
6. Ajouter une méthode `get` retournant l'état d'une cellule (x, y).
7. Ajouter une méthode `toggle` inversant l'état d'une cellule (x, y).
8. Ajouter une méthode `dessine` à la grille. On représentera une case par un carré de z pixels (une constante) de coté.
9. Dans le `main.cpp`, ouvrir une fenêtre de taille $zw \times zh$ (choisir par exemple $w = 100$, $h = 60$) et laisser cliquer l'utilisateur sur des cases. Chaque clic inverse l'état de la cellule et réaffiche la grille. Cette procédure stoppe quand c'est le bouton droit de la souris qui est utilisé.

FIGURE 1 – Gauche : trois formes dont l'évolution est périodique (ligne, grenouille et planeur). Milieu : un état initial et son évolution après un nombre impair d'itérations (noter l'aspect changeant des formes et le déplacement du planeur). Droite : Un texte initial, sa quantification sous forme de cellules (état initial), et son évolution après une centaine d'itérations.

1.3 Le jeu de la vie

10. Pour gérer l'évolution, on ne peut pas le faire en une seule passe sur la grille. Ajouter donc à `Cellule` une deuxième variable booléenne indiquant si son état doit changer.
11. Ajouter à `Grille` une méthode `reset` indiquant qu'a priori aucune cellule ne doit changer détat.
12. Par mesure de sécurité, ajouter des `assert` à toutes les méthodes prenant des coordonnées (x, y).
13. Ajouter une méthode `compte_voisins` comptant le nombre de cases vivantes parmi les 8 voisines d'une cellule (x, y) (attention aux bords de la grille, ces cellules ont moins de 8 voisines).
14. Ajouter une méthode `update` à la grille qui dans une première passe marque les cellules devant changer d'état suivant les règles énoncées en introduction, puis applique ces changements dans une deuxième passe.
15. Écrire une fonction `animation` qui fait évoluer la grille suivant un nombre d'itérations passé en paramètre, avec affichage à chaque itération. L'appeler dans le `main`.
16. Pour gérer les bords, laisser l'option à l'utilisateur de considérer la grille comme un tore : `update` prend un paramètre booléen `tore` qui est utilisé dans `compte_voisins`, et qui suppose que le voisin du dessus d'une cellule de la première ligne est la cellule de la dernière ligne à la même colonne, et de façon similaire pour les trois autres bords.

1.4 Des états initiaux intéressants

17. Ajouter une méthode `ligne` dans `Grille` qui met trois cellules successives verticales (commençant en (x, y)) à l'état vivant. Son évolution sera de façon périodique une barre horizontale et verticale.
18. Ajouter une méthode `grenouille` symbolisant une grenouille (6 cellules), qui oscille également entre deux formes (elle "saute" sur place).
19. Ajouter une méthode `planeur` (5 cellules), figure qui oscille entre deux formes en se déplaçant suivant la diagonale.
20. Faire évoluer une grille avec une ligne, une grenouille et un planeur placés initialement.
21. Créer une fonction `bonneAnnee` qui part d'un état initial avec une version pixellisée du texte. Pour cela, afficher le message avec une taille de police suffisamment grande (40 pixels par exemple) et utiliser la fonction `captureWindow` d'Imagine++ pour récupérer sous forme d'image le texte.
22. Toute cellule dont un des pixels de la zone contient du noir (intersecte le texte) est initialisée vivante.
23. Afficher l'état de la grille résultant (version grossière du texte).
24. Faire évoluer ce texte.

Important : Quand vous avez terminé, créez une archive du projet *à votre nom et numéro de groupe* en ZIP, RAR, TGZ ou 7z, par exemple `G1_Nom_Prenom_Final.zip`. N'incluez que les fichiers source et le CMakeLists.txt, pas les fichiers binaires créés par Cmake. Ne partez pas avant que le surveillant ne soit passé vous voir pour copier cette archive sur clé USB ou avoir pu télécharger votre archive sur Educnet.