TP - Jeu de la Vie et images

Objectif(s)

- ✯ Manipulation de tableaux 2D (images)
- ✯ Révision des algorithmes de tableaux

Exercices obligatoires

Exercice 1 – Jeu de la vie

Dans cette première partie, nous allons programme un jeu de la vie, tel que défini par Conway (cf `https://fr.wikipedia.org/wiki/Jeu_de_la_vie`). Il s'agit d'un automate cellulaire, autant dire une grille avec des cases (cellules) dont la couleur change tour par tour en fonction des couleurs des cases voisines. Dans le cas du jeu de la vie, il n'existe que deux couleurs : vivant et mort.

Dans le jeu de la vie, chaque cellule a huit cellules voisines, les cellules qui partagent un côté ou un sommet. À chaque tour, une cellule vivante reste vivante si elle a deux ou trois cellules voisines vivantes, meurt sinon. Une cellule morte qui a exactement trois cellules voisines vivantes devient vivante à son tour. Dans tous les autres cas, la cellule reste morte.

On utilisera des tableaux de caractères à deux dimensions de dimensions `DIML * DIMC`, deux constantes définies en en-tête (par exemple de largeur 80 et de hauteur 30 mais 3×3 suffit pour les tests).

Question 1

Écrivez une fonction `generer` qui remplit au hasard les cellules avec les caractères choisis pour vivant (par exemple `'o'`) et pour mort (par exemple `' '`).

Question 2

Écrivez une fonction `afficher` qui affiche le tableau des cellules.

Question 3

Écrire et tester une fonction `nb_vivantes` qui retourne le nombre de cellules vivantes parmi les cellules voisines à (i, j), les cases "en dehors" du tableau comptant pour des cellules mortes.

Question 4

Écrire une fonction `copie` qui prends deux tableaux de caractères de même taille en paramètre et copie le premier dans le second.

Question 5

Écrire une fonction `suivant` qui calcule la configuration au tour suivant. Comme ce calcul modifie la configuration, il est indispensable de passer par un deuxième tableau, vous pourrez utiliser la fonction `copie` de la question précédente (à bon escient).

Question 6

Écrire une fonction `afficher_jeu` qui, étant donné une configuration initiale et un nombre n de tours, affiche les états successifs du tableau. Lorsque votre programme fonctionnera correctement, nous vous proposons d'ajouter `usleep(500000);` juste après l'affichage d'un tableau suivi de `printf("\033[2J");` juste avant l'affichage du tableau suivant, ce qui effacera votre terminal en positionnant le curseur en haut à gauche, et réalisera une petite animation.

Question 7

Écrivez une version du jeu, où on supposera que les cellules de la première ligne du tableau sont voisines des cellules de la dernière ligne, et que les cellules de la première et dernière colonnes du tableau sont aussi voisine. Ainsi, le tableau est périodique dans les deux directions, comme sur un donut (ou un tore), et fait vivre les cellules dans un plan infini.

Pour éviter d'avoir des résultats de modulo négatif pour la première ligne et la la première colonne, vous pouvez vous inspirer de la formule suivante : `(DIML + i - 1)% DIML` pour la ligne précédente de la première ligne à traiter.

Renforcements

Exercice 2 – Plus d'images

Question 1

Écrivez une fonction `noirEtBlanc(img)` qui convertit une image monochrome (telle que produite par la fonction `monochrome` de la question **??**) en une image noir et blanc : chaque pixel peut valoir (0,0,0) ou (255,255,255)) selon que la luminosité est plus petite ou plus grande que 127.

Question 2

Comme vous l'avez constaté, la qualité des images produites par la fonction précédente laisse à désirer. L'algorithme proposé par Floyd et Steinberg permet de limiter la perte d'information due à la quantification des pixels en blanc ou noir. Écrivez une fonction `floydSteinberg(img)` qui convertit une image monochrome en noir et blanc à l'aide de l'algorithme de Floyd et Steinberg (cf. page Wikipedia : `http://fr.wikipedia.org/wiki/Algorithme_de_Floyd-Steinberg`).

Question 3

Agrandissement par interpolation

Nous allons voir maintenant une méthode permettant des agrandissements d'un facteur rationnel $F \geq 1$.

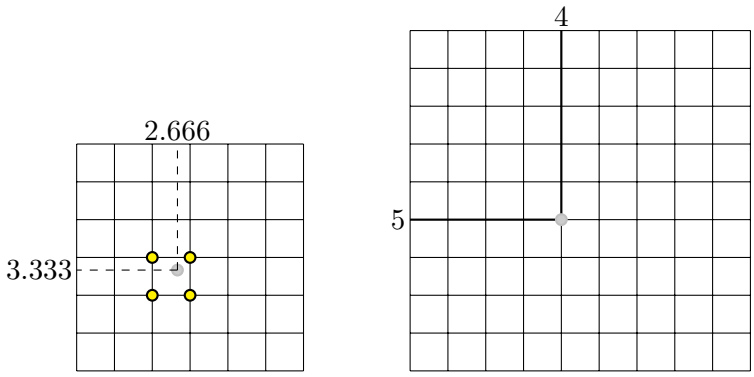

FIGURE 1 – Agrandissement d'un facteur $3/2$ d'une image 7×7

La Figure 1 schématise l'agrandissement par un facteur $F = 3/2$ d'une image source de 7×7 pixels. L'agrandissement aura pour dimensions 10×10 : on ne peut pas faire $10,5 \times 10,5$ pixels.

Dans l'agrandissement, le pixel de coordonnées $(4, 5)$ est le fruit d'un mélange des quatre pixels encadrant dans l'image source le point $(4/\frac{3}{2}, 5/\frac{3}{2})$ = $(\frac{8}{3}, \frac{10}{3})$ = $(2 + \frac{2}{3}, 3 + \frac{1}{3})$. Les proportions du mélange sont calculées de façon à accorder une part prépondérante au(x) pixel(s) le(s) plus proche(s). La Figure 2 illustre la technique que nous allons mettre en œuvre : le cœfficient d'un pixel sera égale à la surface du rectangle opposé. Dans notre exemple, illustré à gauche de la figure 2, la couleur de notre pixel $(4, 5)$ sera donc composé de $4/9$ du pixel (3,3) (pixel le plus proche du point), de $2/9$ des pixels (2,3) et (3,4), et de $1/9$ du pixel (2,4).

De façon générale, illustrée à droite de la figure 2, pour déterminer les pixels contribuant au pixel p_{ij}, on calcule sa position virtuelle (x, y) dans l'image source en divisant les indices par le facteur d'agrandissement. On obtient ainsi un point de coordonnées réelles $(x, y) = (i/F, j/F)$. Les coordonnées des pixels voisins de ce point sont alors formés

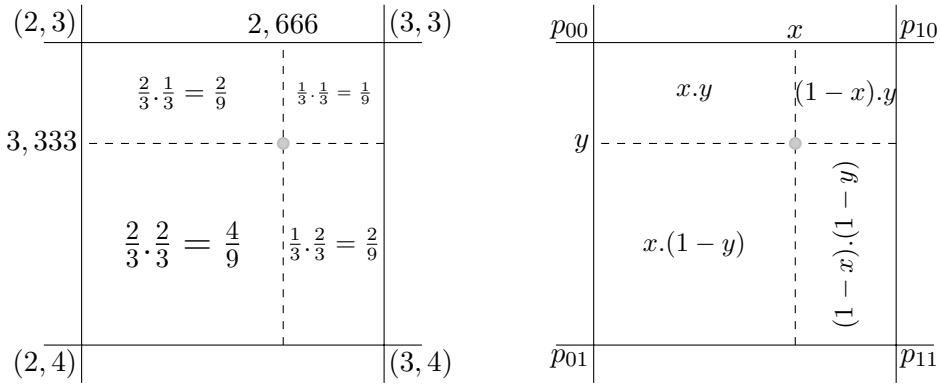

FIGURE 2 – Proportions du mélange

par les parties entières par défaut et par excès de x et y : p_{00} en $(\lfloor x \rfloor, \lfloor y \rfloor)$ p_{01} en $(\lfloor x \rfloor, \lceil y \rceil)$ p_{10} en $(\lceil x \rceil, \lfloor y \rfloor)$ et p_{11} en $(\lceil x \rceil, \lceil y \rceil)$.

Pour calculer l'apport des pixels sources, on procède à un changement de repère pour positionner l'origine en p_{00}, les coordonnées x et y sont alors comprises entre 0 et 1. Le calcul des aires des rectangles est alors aisé comme le montre la partie droite de la Figure 2. Voici le code de la fonction calculant cette quantité :

```
int interpolation_bilineaire(int p00, int p01, int p10, int p11, double x,
 ↪ double y){
    double xf = x - (int) x;
    double yf = y - (int) y;
    return ( (1. - xf) * (1. - yf) * p00 + (1. - xf) * yf * p01
                + xf * (1. - yf) * p10 + xf * yf * p11) ;
}
```

Pour utiliser cette fonction on placera dans les paramètres `p00, p01, p10, p11` les valeurs d'une même composante rvb des pixels voisins et dans `x,y` les coordonnées réelles du pixel destination projeté dans l'image source avec le facteur F.

Écrire une fonction `agrandir(img,facteur)` qui retourne une image correspondant à l'agrandissement de l'image `img` en utilisant la technique d'interpolation bilinéaire pour calculer chaque composante rvb de chaque pixel de l'agrandissement.