

```

1  /*
2  -----
3  Laboratoire : 05
4  Fichier      : jeux_de_la_vie.cpp
5  Auteur(s)    : BOUSBAA Eric, BOTTIN Stéphane, FERRARI Teo
6  Date         : 10.01.2019
7
8  But          : Met à disposition des fonctions ayant pour but de simuler le jeu de
9                 la vie.
10
11  Remarque(s) : - Les valeurs définissant si une cellule doit naître ou survivre sont
12                 défini dans des listes constantes. Chaque valeur de la liste
13                 représente une condition à laquelle la règle associé est valide.
14                 - Afin de modifier la configuration initiale du tableau de base,
15                 il suffit de placer un 1 dans toute cellule voulant être
16                 initialement habitée.
17                 - La taille du tableau initial peut être changé en ajoutant/supprimant
18                 les lignes/colonnes dans son initialisation.
19                 - Toutes les altération d'une génération de cellules à une autre
20                 s'effectuent simultanément. Càd qu'il n'y a pas d'ordre de priorité
21                 entre vivre et mourir.
22
23  Compilateur : - Apple LLVM version 9.0.0 (clang-900.0.39.2)
24                 - MinGW-g++ 6.3.0
25  -----
26  */
27
28  #include <cstdlib>
29  #include <iostream>
30  #include <algorithm>
31  #include "jeux_de_la_vie.h"
32
33  using namespace std;
34
35  const char CARACTERE_VIE = 'X';
36  const char CARACTERE_MORT = '.';
37
38  /**
39   * @brief Affiche le tableau représentant le jeu de la vie en utilisant des
40   * caractères pour représenter les états de vie et de mort des cellules
41   * @param tableau : tableau booléen à afficher
42   * @param caractereVie : caractere représentant un cellule
43   * @param caractereMort : caractere représentant un absence de cellule
44   */
45  void afficherTableau(const vector<vector<bool>>&tableau,
46                      char caractereVie, char caractereMort);
47
48  /**
49   * @brief fonction qui retourne l'état futur d'une case spécifique du tableau
50   * représentant le jeu de la vie
51   * @param tableau : tableau booléen représentant le jeu de la vie
52   * @param i : position verticale de la case à tester
53   * @param j : position horizontale de la case à tester
54   * @return : l'état dans lequel devrait être la case par rapport au cases l'entourant
55   */
56  bool etatFutur(const vector<vector<bool>>&tableau, unsigned i, unsigned j);
57
58  /**
59   * @brief fonction vérifiant les occurrences de cellules autour d'une cellule donnée
60   * @param tableau : tableau booléen représentant le jeu de la vie
61   * @param x : position horizontale de la case à tester
62   * @param y : position verticale de la case à tester
63   * @return le nombre d'occurrences de cellules autour de la cellule donnée
64   */
65  unsigned occ(const vector<vector<bool>>&tableau, unsigned x, unsigned y);
66
67  /**
68   * @brief vérifie qu'une valeur est dans un intervalle donnée
69   * @param V : Liste de valeurs devant être vérifiés
70   * @param val : valeur à trouver dans l'intervalle
71   * @return si la valeur se trouve dans l'intervalle ou non
72   */
73  bool estDansIntervalle(const vector<unsigned> &V, const unsigned val);
74
75  const unsigned NOMBRE_CASES_VOISINES = 8;
76
77  void simulation(unsigned nombreSimulations) {
78
79      vector<vector<bool>> tableauPresent = {
80          {0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0},
81          {0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0},
82          {1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0},
83          {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
84          {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
85          {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
86          {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

```

```

87         {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
88         {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
89         {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
90     };
91
92     vector <vector<bool>> tableauFutur(tableauPresent.size(),
93         vector<bool>(tableauPresent[0].size()));
94
95     for (unsigned n = 1; n <= nombreSimulations; ++n) {
96
97         cout << "Generation : " << n << endl;
98         afficherTableau(tableauPresent, CARACTERE_VIE, CARACTERE_MORT);
99         cout << endl;
100
101         for (unsigned i = 0; i < tableauPresent.size(); ++i) {
102             for (unsigned j = 0; j < tableauPresent[0].size(); ++j) {
103                 tableauFutur[i][j] = etatFutur(tableauPresent, i, j);
104             }
105         }
106         tableauPresent = tableauFutur;
107     }
108 }
109
110 unsigned occ(const vector < vector<bool>>&tableau, unsigned x, unsigned y) {
111     unsigned nbCasesVoisines = 0;
112     int positionAbsolueX, positionAbsolueY;
113     // Tableau d'indices autour d'une cellule, permettant d'accéder au contenu de celles-ci
114     int positionRelativeX[] = {1, -1, 0, 0, 1, 1, -1, -1};
115     int positionRelativeY[] = {0, 0, 1, -1, 1, -1, 1, -1};
116
117     for (int i = 0; i < (int) NOMBRE_CASES_VOISINES; ++i) {
118         // Afin d'obtenir les réelles positions autour de la cellule à calculer,
119         // il suffit d'additionner la position relative avec la relation courante.
120         positionAbsolueX = (int) x + positionRelativeX[i];
121         positionAbsolueY = (int) y + positionRelativeY[i];
122         // Les cellules hors de la grille ne sont pas vérifiées
123         if (positionAbsolueX < (int) tableau[0].size() && positionAbsolueX >= 0 &&
124             positionAbsolueY >= 0 && positionAbsolueY < (int) tableau.size()) {
125             if (tableau[(size_t) positionAbsolueY][(size_t) positionAbsolueX] == 1) {
126                 nbCasesVoisines++;
127             }
128         }
129     }
130     return nbCasesVoisines;
131 }
132
133 void afficherTableau(const vector <vector<bool>>&tableau, char caractereVie,
134     char caractereMort) {
135     for (size_t i = 0; i < tableau.size(); ++i) {
136         for (size_t j = 0; j < tableau[0].size(); ++j) {
137             cout << " " << (tableau[i][j] ? caractereVie : caractereMort) << " ";
138         }
139         cout << endl;
140     }
141 }
142
143 bool estDansIntervalle(const vector<unsigned> &V, const unsigned val) {
144     for (auto i = V.begin(); i != V.end(); ++i) {
145         if (val == *i) {
146             return true;
147         }
148     }
149     return false;
150 }
151
152 bool etatFutur(const vector <vector<bool>>&tableau, unsigned i, unsigned j) {
153     unsigned occurrences = occ(tableau, j, i);
154     if (tableau[i][j]) {
155         return estDansIntervalle(REGLE_NAISSANCE, occurrences) ||
156             estDansIntervalle(REGLE_SURVIS, occurrences) ? 1 : 0;
157     } else {
158         return estDansIntervalle(REGLE_NAISSANCE, occurrences) ? 1 : 0;
159     }
160 }
161

```