```cpp
/*
 ------------------------------------------------------------------------------
 Laboratoire : 05
 Fichier     : jeux_de_la_vie.cpp
 Auteur(s)   : BOUSBAA Eric, BOTTIN Stéphane, FERRARI Teo
 Date        : 10.01.2019

 But         : Met à disposition des fonctions ayant pour but de simuler le jeux de
               la vie.

 Remarque(s) : - Les valeurs définissant si une cellule doit naître ou survivre sont
                 défini dans des listes constantes. Chaque valeur de la liste
                 représente une condition à laquelle la règle associé est valide.
               - Afin de modifier la configuration initiale du tableau de base,
                 il suffit de placer un 1 dans toute cellule voulant être
                 initialement habitée.
               - La taille du tableau initial peut être changé en ajoutant/supprimant
                 les lignes/colonnes dans son initialisation.
               - Toutes les altération d'une génération de cellules à une autre
                 s'effectuent simultanément. Càd qu'il n'y a pas d'ordre de priorité
                 entre vivre et mourir.

 Compilateur : - Apple LLVM version 9.0.0 (clang-900.0.39.2)
               - MinGW-g++ 6.3.0
 ------------------------------------------------------------------------------
 */

#include <cstdlib>
#include <iostream>
#include <algorithm>
#include "jeux_de_la_vie.h"

using namespace std;

const char CARACTERE_VIE = 'X';
const char CARACTERE_MORT = '.';

/**
 * @brief Affiche le tableau représentant le jeu de la vie en utilisant des
 * charactères pour représenter les états de vie et de mort des cellules
 * @param tableau : tableau booleen a afficher
 * @param caractereVie : caraectere representant un cellule
 * @param caractereMort : caraectere representant un abscence de cellule
 */
void afficherTableau(const vector<vector<bool>>&tableau,
        char caractereVie, char caractereMort);

/**
 * @brief fonction qui retourne l'état futur d'une case specifique du tableau
 * representant le jeu de la vie
 * @param tableau : tableau booleen representant le jeu de la vie
 * @param i : position verticale de la case a tester
 * @param j : position horizontale de la case a tester
 * @return : l'etat dans lequel devrait etre la case par rapport au cases l'entourant
 */
bool etatFutur(const vector<vector<bool>>&tableau, unsigned i, unsigned j);

/**
 * @brief fonction verifiant les occurences de cellules autour d'une cellule donnee
 * @param tableau : tableau booleen representant le jeu de la vie
 * @param x : position horizontale de la case a tester
 * @param y : position verticale de la case a tester
 * @return le nombre d'occurences de cellules autour de la cellule donnee
 */
unsigned occ(const vector <vector<bool>>&tableau, unsigned x, unsigned y);

/**
 * @brief verifies qu'une valeur est dans un intervalle donnée
 * @param V : Liste de valeurs devant être vérifiés
 * @param val : valeur à trouver dans l'intervalle
 * @return si la valeur se trouve dans l'intervalle ou non
 */
bool estDansIntervalle(const vector<unsigned> &V, const unsigned val);

const unsigned NOMBRE_CASES_VOISINES = 8;

void simulation(unsigned nombreSimulations) {

   vector <vector<bool>> tableauPresent = {
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0},
      {0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
```

```
87          {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
88          {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
89          {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, postitionAbsolueY;
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         postitionAbsolueY = (int) y + positionRelativeY[i];
122         // Les cellules hors de la grille ne sont pas vérifiés
123         if (positionAbsolueX < (int) tableau[0].size() && positionAbsolueX >= 0 &&
124                postitionAbsolueY >= 0 && postitionAbsolueY < (int) tableau.size()) {
125            if (tableau[(size_t) postitionAbsolueY][(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 occurences = occ(tableau, j, i);
154      if (tableau[i][j]) {
155         return estDansIntervalle(REGLE_NAISSANCE, occurences) ||
156                 estDansIntervalle(REGLE_SURVIS, occurences) ? 1 : 0;
157      } else {
158         return estDansIntervalle(REGLE_NAISSANCE, occurences) ? 1 : 0;
159      }
160   }
161
```