```c
//------------------------------------------------------------------------------
// sciper: 253133
// nom: Borden
// prenom: Sven
// fichier: conway.c
// date: 12.10.2015
// description du programme: projet PROG I Automne 2015-16 EPFL MT-EL
//------------------------------------------------------------------------------

#include <stdio.h>
#include <stdlib.h>

#define ZOOM_MAX    100
#define ZOOM_MIN    1
#define SIZE_FORMAT 2
#define NB_SAVE     3
#define MIN_NB_SAVE 2
#define ALIVE       1
#define DEAD        0
#define CHARMAX     34

#define CENTRE      0
#define BAS         1
#define GAUCHE      2
#define BAS_GAUCHE  3
#define DROITE      4
#define BAS_DROITE  5
#define HAUT        8
#define HAUT_GAUCHE 10
#define HAUT_DROITE 12

struct basicVal
{
    int ligne;
    int colonne;
    int zoom;
    int nbJ;
    int nbS;
};

//------------------------------------------------------------------------------
//                 NE PAS MODIFIER CES PROTOTYPES DE FONCTIONS

static void erreur_nbJ(int nbJ);
static void erreur_nbS(int nbS);
static void erreur_nbJ_nbS(int nbJ, int nbS);
static void erreur_zoom(int zoom);
//------------------------------------------------------------------------------

static void lecture();
static void analyse(int* pT1, int* pT2, struct basicVal* pVal);
static int caseVivante(int* tab, int position, struct basicVal* pVal, int cas);
static void start(int* pT1, int* pT2, struct basicVal* pVal);
static void output(int* tableau, struct basicVal* pVal, int compteur);
static void header(struct basicVal*);
static void reprint(int question);

static int verbose = 0;

int main(int argc, const char* argv[])
{
    lecture();
    return EXIT_SUCCESS;
}

static void lecture()
{
    int i, j;
    int question = 0;
    char format[SIZE_FORMAT];
    struct basicVal val;
    struct basicVal *pVal = &val;

```

```c
    scanf("%d", &verbose);

    reprint(question++);
    scanf("%d", &val.nbJ);
    if (val.nbJ < 0)
        erreur_nbJ(val.nbJ);

    reprint(question++);
    scanf("%d", &val.nbS);
    if (val.nbS < 0)
        erreur_nbS(val.nbS);
    if ((val.nbS > 0 && val.nbJ % val.nbS != 0) || (val.nbJ == 0 && val.nbS > 1))
        erreur_nbJ_nbS(val.nbJ, val.nbS);

    reprint(question++);
    scanf("%d", &val.zoom);
    if (val.zoom < ZOOM_MIN || val.zoom > ZOOM_MAX)
        erreur_zoom(val.zoom);

    reprint(question++);
    scanf("%s", format);

    reprint(question++);
    scanf("%d", &val.colonne);
    scanf("%d", &val.ligne);

    reprint(question++);

    int tabInit[val.ligne][val.colonne];
    int tabSuiv[val.ligne][val.colonne];
    int *pTabInit = (int*)tabInit;
    int *pTabSuiv = (int*)tabSuiv;
    for (i = 0; i < val.ligne; i++)
        for (j = 0; j < val.colonne; j++)
            scanf("%d", &tabInit[i][j]);

    if (val.nbS != 0)
    {
        header(pVal);
        output(pTabInit, pVal, 0);
    }
    start(pTabInit, pTabSuiv, pVal);
}

static void analyse(int* pT1, int* pT2, struct basicVal* pVal)
{
    int i, posX, posY;
    int ligne = pVal->ligne;
    int colonne = pVal->colonne;
    for (i = 0; i < ligne * colonne; i++)
    {
        posX = i / colonne;
        posY = i % colonne;

        if ((posX > 0) && (posY > 0) && (posX < ligne-1) && (posY < colonne-1))
            *(pT2 + i) = caseVivante(pT1, i, pVal, CENTRE);
        else
        {
            if ((posX == ligne - 1) &&
                (posY > 0) && (posY < colonne - 1))
                *(pT2 + i) = caseVivante(pT1, i, pVal, BAS);

            if ((posX > 0) && (posX < ligne - 1) &&
                (posY == 0))
                *(pT2 + i) = caseVivante(pT1, i, pVal, GAUCHE);

            if ((posX == ligne - 1) &&
                (posY == 0))
                *(pT2 + i) = caseVivante(pT1, i, pVal, BAS_GAUCHE);

            if ((posX > 0) && (posX < ligne - 1) &&
                (posY == colonne - 1))
                *(pT2 + i) = caseVivante(pT1, i, pVal, DROITE);
```

```
147
148             if ((posX == ligne - 1) &&
149                 (posY == colonne - 1))
150                 *(pT2 + i) = caseVivante(pT1, i, pVal, BAS_DROITE);
151
152             if ((posX == 0) &&
153                 (posY > 0) && (posY < colonne - 1))
154                 *(pT2 + i) = caseVivante(pT1, i, pVal, HAUT);
155
156             if ((posX == 0) &&
157                 (posY == 0))
158                 *(pT2 + i) = caseVivante(pT1, i, pVal, HAUT_GAUCHE);
159
160             if ((posX == 0) &&
161                 (posY == colonne - 1))
162                 *(pT2 + i) = caseVivante(pT1, i, pVal, HAUT_DROITE);
163         }
164
165     }
166 }
167
168 static int caseVivante(int* tab, int position, struct basicVal* pVal, int cas)
169 {
170     int count = 0;
171     int colonne = pVal->colonne;
172     tab += position;
173
174     switch (cas)
175     {
176     case CENTRE:
177         count += *(tab - 1 - colonne);
178         count += *(tab - colonne);
179         count += *(tab + 1 - colonne);
180         count += *(tab - 1);
181         count += *(tab + 1);
182         count += *(tab - 1 + colonne);
183         count += *(tab + colonne);
184         count += *(tab + 1 + colonne);
185         break;
186     case BAS:
187         count += *(tab - 1 - colonne);
188         count += *(tab - colonne);
189         count += *(tab + 1 - colonne);
190         count += *(tab - 1);
191         count += *(tab + 1);
192         break;
193     case GAUCHE:
194         count += *(tab - colonne);
195         count += *(tab + 1 - colonne);
196         count += *(tab + 1);
197         count += *(tab + colonne);
198         count += *(tab + 1 + colonne);
199         break;
200     case BAS_GAUCHE:
201         count += *(tab - colonne);
202         count += *(tab + 1 - colonne);
203         count += *(tab + 1);
204         break;
205     case DROITE:
206         count += *(tab - 1 - colonne);
207         count += *(tab - colonne);
208         count += *(tab - 1);
209         count += *(tab - 1 + colonne);
210         count += *(tab + colonne);
211         break;
212     case BAS_DROITE:
213         count += *(tab - 1 - colonne);
214         count += *(tab - colonne);
215         count += *(tab - 1);
216         break;
217     case HAUT:
218         count += *(tab - 1);
219         count += *(tab + 1);
```

```
220             count += *(tab - 1 + colonne);
221             count += *(tab + colonne);
222             count += *(tab + 1 + colonne);
223             break;
224         case HAUT_GAUCHE:
225             count += *(tab + 1);
226             count += *(tab + colonne);
227             count += *(tab + 1 + colonne);
228             break;
229         case HAUT_DROITE:
230             count += *(tab - 1);
231             count += *(tab - 1 + colonne);
232             count += *(tab + colonne);
233             break;
234         default:
235             break;
236         }
237         if (*tab != ALIVE)
238             return (count == NB_SAVE) ? ALIVE : DEAD;
239         else
240             return (count == MIN_NB_SAVE || count == NB_SAVE) ? ALIVE : DEAD;
241     }
242
243     static void start(int* pT1, int* pT2, struct basicVal* pVal)
244     {
245         int i;
246         for (i = 1; i <= pVal->nbJ; i++)
247         {
248             if (i % 2 == 0)
249                 analyse(pT2, pT1, pVal);
250             else
251                 analyse(pT1, pT2, pVal);
252
253             if (pVal->nbS != 0 && i % pVal->nbS == 0)
254                 if (i % 2 == 0)
255                     output(pT1, pVal, i);
256                 else
257                     output(pT2, pVal, i);
258         }
259     }
260
261     static void output(int* tableau, struct basicVal* pVal, int compteur)
262     {
263         int i, j, k, l, charcompteur = 0;
264         if (compteur != 0) //imprime une ligne noire
265             for (i = 0; i < pVal->colonne * pVal->zoom; i++, charcompteur++)
266             {
267                 if (charcompteur == CHARMAX)
268                 {
269                     printf("\n");
270                     charcompteur = 0;
271                 }
272                 printf("1 ");
273             }
274         printf("\n");
275         charcompteur = 0;
276
277         //impression du tableau
278         for (i = 0; i < pVal->ligne; i++)
279             for (k = 0; k < pVal->zoom; k++)
280             {
281                 for (j = 0; j < pVal->colonne; j++)
282                     for (l = 0; l < pVal->zoom; l++, charcompteur++)
283                     {
284                         if (charcompteur == CHARMAX)
285                         {
286                             printf("\n");
287                             charcompteur = 0;
288                         }
289                         printf("%d ", *(tableau + (i*pVal->colonne) + j));
290                     }
291                 printf("\n");
292                 charcompteur = 0;
```

```c
        }
}

static void header(struct basicVal * pVal)
{
    int col = pVal->colonne * pVal->zoom;
    int lig = pVal->ligne * pVal->zoom;
    lig += (pVal->nbJ / pVal->nbS) * (pVal->ligne * pVal->zoom);
    lig += pVal->nbJ / pVal->nbS;
    printf("P1\n%d %d\n", col, lig);
}

static void reprint(int question)
{
    if (verbose)
        switch (question)
        {
        case 0: printf("Entrez le nombre de mises à jours\n");
            break;
        case 1: printf("Entrez la période d'affichage\n");
            break;
        case 2: printf("Entrez le facteur de zoom\n");
            break;
        case 3: printf("Entrez le code du format de l'image\n");
            break;
        case 4: printf("Entrez les dimensions de l'image\n");
            break;
        case 5: printf("Entrez les valeurs du monde\n");
            break;
        default: break;
        }
}

//------------------------------------------------------------------------------
// Fonctions prédéfinies pour indiquer si les données sont correctes
// Les fonctions signalant une erreur provoquent la fin du programme
// en appelant exit(). Leur message d'erreur est toujours affiché.
//
//                 NE PAS MODIFIER CES FONCTIONS
//------------------------------------------------------------------------------

// A appeler si le nombre de mises à jours n'est pas positif
static void erreur_nbJ(int nbJ)
{
    printf("Le nombre de mises à jours nbJ n'est pas positif: %d\n", nbJ);
    exit(EXIT_FAILURE);
}

// A appeler si la periode de sauvegarde n'est pas positive
static void erreur_nbS(int nbS)
{
    printf("La période de sauvegarde nbS n'est pas positive: %d\n", nbS);
    exit(EXIT_FAILURE);
}

// A appeler si la periode de sauvegarde n'est pas positive
static void erreur_nbJ_nbS(int nbJ, int nbS)
{
    printf("La combinaison du nombre de mises à jour %d et de la période de "
        "sauvegarde %d est interdite\n", nbJ, nbS);
    exit(EXIT_FAILURE);
}

// A appeler si le facteur de zoom n'est pas dans le bon intervalle
static void erreur_zoom(int zoom)
{
    printf("Le facteur de zoom %d n'est pas compris dans l'intervalle ]0,100]\n",
        zoom);
    exit(EXIT_FAILURE);
}
```