Algorithmique & Programmation

Le jeu de la vie

ou le microcosme des automates cellulaires

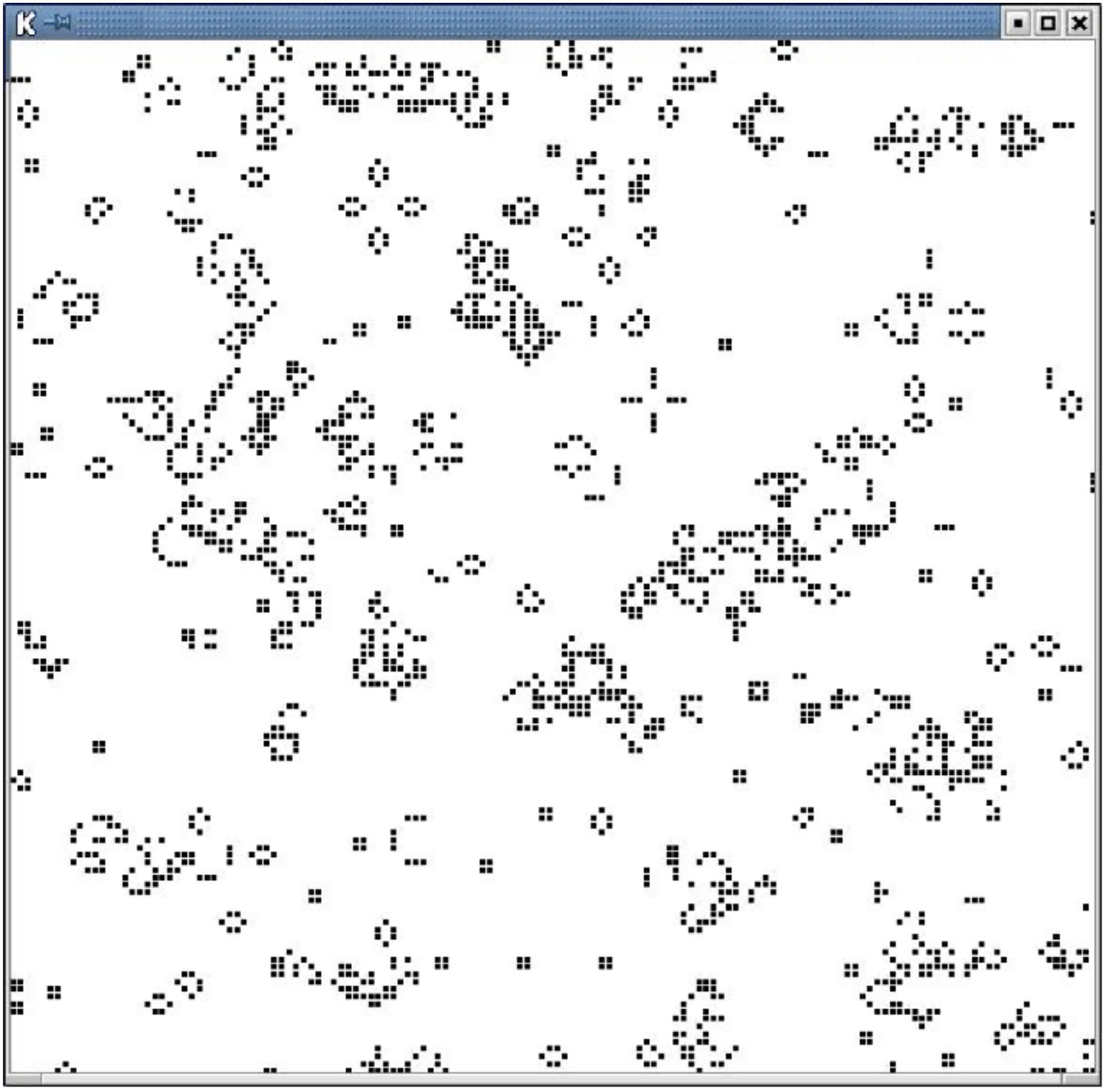
yann.secq@univ-lille.fr

ABIDI Sofiene, ALMEIDA COCO Amadeu, BONEVA Iovka, CASTILLON Antoine, DELECROIX Fabien, LEPRETRE Éric, Timothé ROUZÉ, SANTANA MAIA Deise, SECQ Yann

Saison 2023-2024 / R1.01+R1.02 :
Initiation au développement

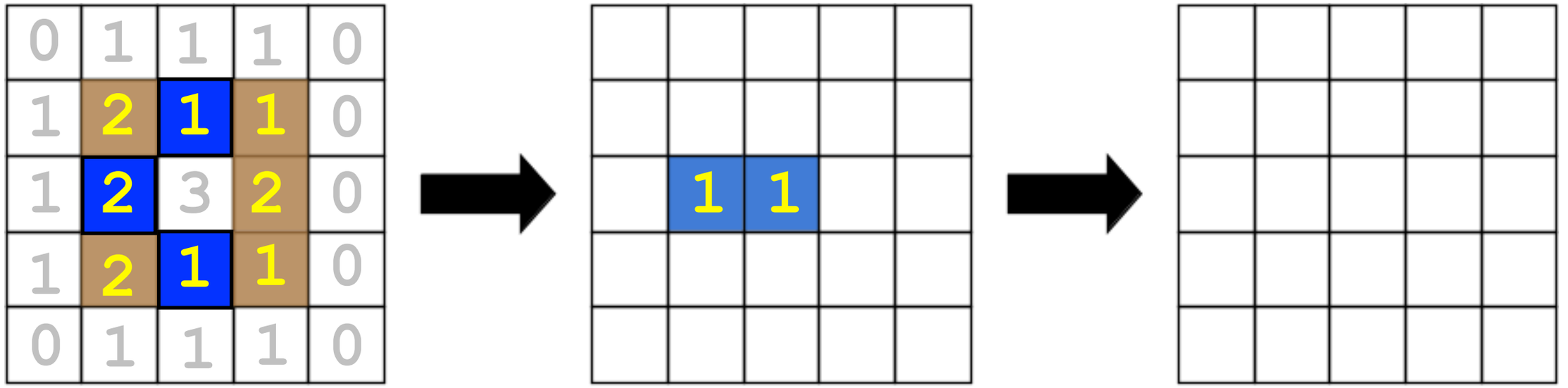
Le jeu de la vie

- Inventé par John H. Conway (1970)
- Pas vraiment un « jeu » :)
- Un plateau, des cellules, quelques règles d'évolution
- Pourquoi tant d'intérêt ?
  - Des formes surprenantes
  - Des propriétés théoriques liées à l'informatique fondamentale !



Règles d'évolution

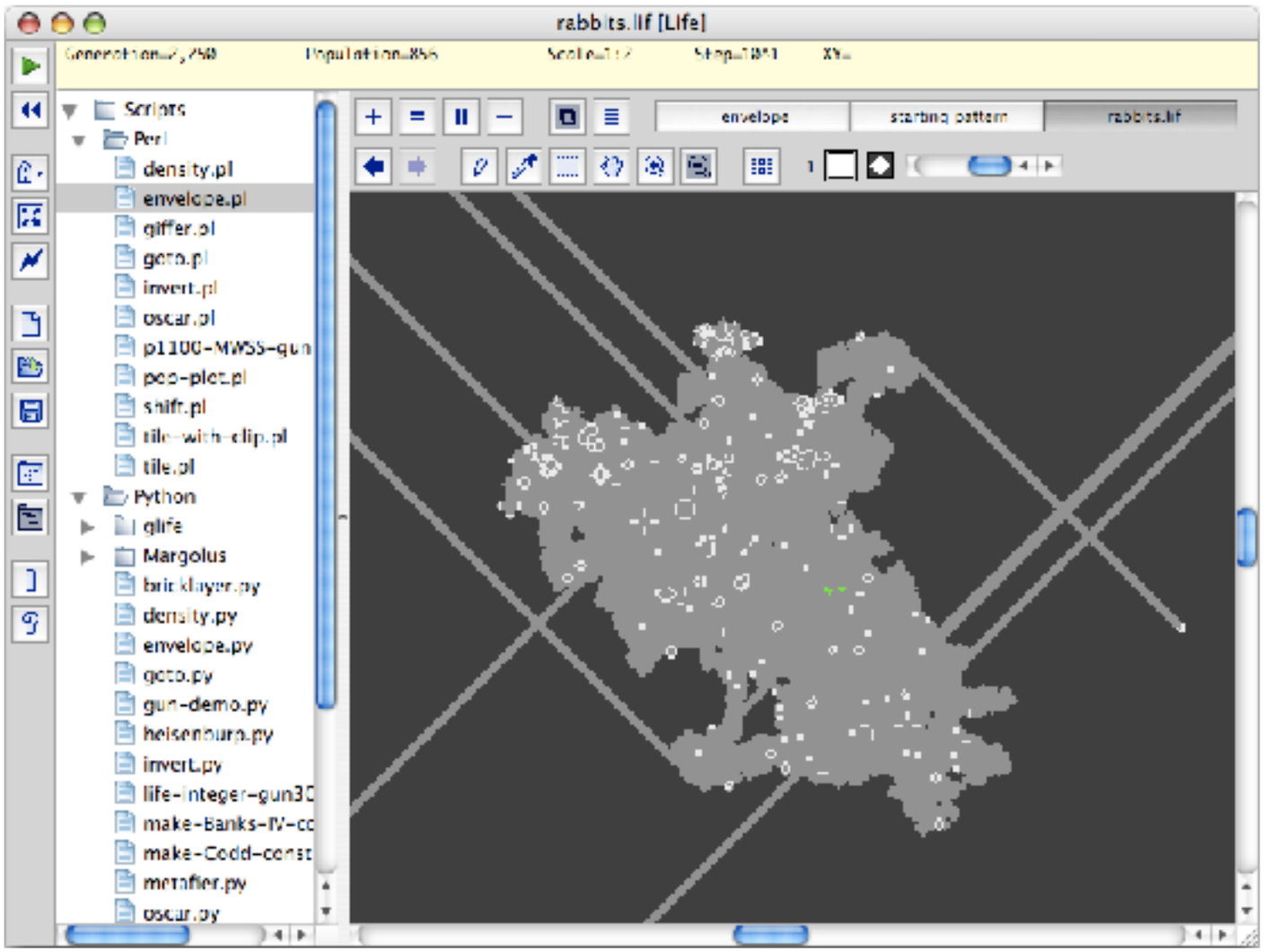
- Un plateau 2D contenant une cellule vivante/morte par case
- Règles d'évolution (liées au voisinage dit de Conway) :
  - **cellule morte** entourée d'exactement trois cellules vivantes ☞ la cellule devient vivante
  - **cellule vivante** entourée de 2 ou 3 cellules vivantes ☞ elle reste vivante, sinon elle meurt
- Evolution simultanée de l'ensemble des cellules


0 1 1 1 0
1 2 1 1 0
1 2 3 2 0
1 2 1 1 0
0 1 1 1 0
1 1
1 2 2 1
2 2 2 1
2 2 3 1
1 1 1 0
3 3
3 3
1 1 1
2 1 2
3 2 3
2 1 2
1 1 1
3
1 2 1
3

Quelle biodiversité !

- Les motifs stables
- Les oscillateurs
- Les glisseurs
- Les vaisseaux
- Les armes
- Les jardins du paradis ...

golly.sf.net



Une vidéo … surprenante !

Analyse du jeu de la vie

- Comment représenter une cellule ?
- Comment représenter le plateau ?
- Comment accéder au voisinage d'une cellule ?
- En quoi consiste un tour de jeu ?
- Présentation de l'information à l'utilisateur ?
- Que peut-on paramétrer dans ce logiciel ?

Structure de données

- Une cellule est soit vivante, soit morte
  - 2 états, donc un booléen
- Deux possibilités pour le plateau de jeu
  - un tableau en 2 dimensions de cellules
  - un tableau de coordonnées des cellules vivantes
- Toujours penser aux opérations les plus complexes à effectuer sur la SDD !

Vue d'ensemble

- Création du monde et de sa population
- Tant que l'utilisateur le veut ou un nombre donné de générations n'est pas atteint
  - Afficher l'état actuel de la population
  - Calcul de la population à l'état suivant

Tour de simulation

- Parcours de l'ensemble des cases du tableau (vraiment toutes les cases ?)
- Pour chaque case, appliquer les règles d'évolution
- Quel problème se pose concernant la mise à jour de l'état des cellules ?

Tour de simulation

- Balayer l'ensemble cellules sauf la première ligne/colonne et dernière ligne/colonne
- Pour chaque cellule de coordonnées `(l, c)`, compter les cellules vivantes dans le voisinage
- Appliquer les règles d'évolution et stocker l'état de la cellule `(l, c)` dans un autre tableau

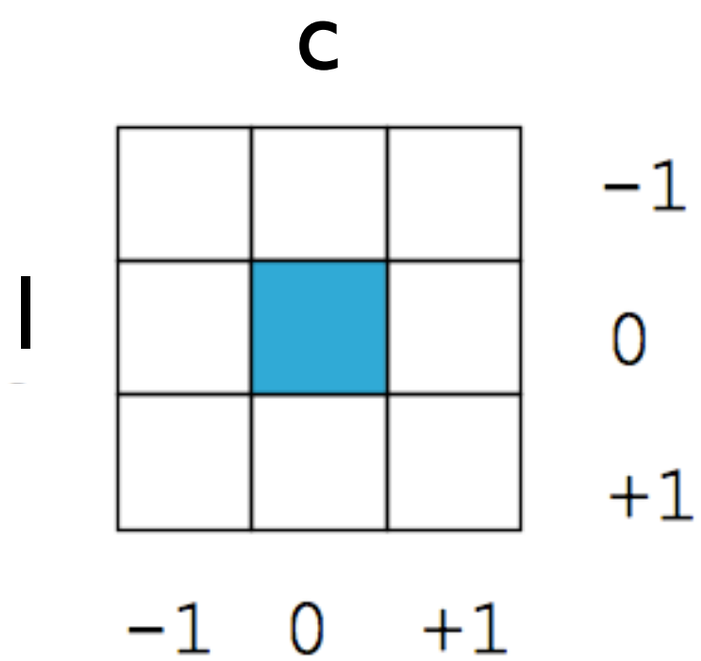
Calcul d'une génération

```
boolean[][] monde, mondeN;

// Boucle sur les lignes
for (int l=1; l<length(monde, 1)-1; l=l+1) {
  // Boucle sur les colonnes
  for (int c=1; c<length(monde, 2)-1; c=c+1) {
    // Calcul du nombre de voisins pour monde[l][c]
    // Calcul de l'evolution dans mondeN
  }
}
// Recopier le tableau mondeN dans monde
println(monde); // ou presque
```

Décompte du voisinage

- Tester les 8 cases adjacentes et cumuler le nombre de cellules vivantes
- Deux boucles imbriquées permettent de balayer l'ensemble des (9!) cases



Nombre de cellules vivantes autour de `(i,j)`

```
int voisins = 0;
// Boucle sur les lignes
for (int decLig=-1; decLig<=1; decLig++) {
  // Boucle sur les colonnes
  for (int decCol=-1; decCol<=1; decCol++) {
    if (m[                                    &&
        (                                  )) {
      voisins = voisins + 1;
    }
  }
}
```

Règles d'évolution

- Pour une case de coordonnée `(l,c)`:

```
if (m[redacted]) {
  mondeN[l][c] = true;
} else if [redacted] {
  mondeN[l][c] = true;
} else {
  mondeN[l][c] = false;
}
```

```
for (int l=1; l<length(monde,1)-1; l++) {
  for (int c=1; c<length(monde,2)-1; c++) {
    voisins = 0;
    for (int decLig=-1; decLig<=1; decLig++) {
      for (int decCol=-1; decCol<=1; decCol++) {
        if ([redacted]
            [redacted]) {
          voisins = voisins + 1;
        }
      }
    }
    if ([redacted]) {
      mondeN[l][c] = true;
    } else if ([redacted]) {
      mondeN[l][c] = true;
    } else {
      mondeN[l][c] = false;
    }
    // Copie de mondeN dans monde
    // Affichage du tableau
  }
}
```

Un peu de génie logiciel …

- Une version monolithique est possible …
- Mais elle est assez peu lisible :(
- Comment rentre plus lisible et plus facile à maintenir et faire évoluer ce programme ?
- La notion de fonction est une première réponse ! (avant la POO ;))
- Comment décomposer le jeu de la vie ?

Démarche pour décomposer

- Identifier les traitements « atomiques »
- Identifier les données nécessaires pour réaliser le traitement et le résultat produit
- Identifier la signature de la fonction à créer
- Déplacer les instructions dans le corps de la fonction et remplacer le code déplacé par l'appel à la fonction créée

```
boolean[][] monde, mondeN; String continuer;
// allocation de monde/N et initialisation de monde
while (!equals(continuer,"n")) {
  // Affichage de monde
  for (int l=1; l<length(monde,1)-1; l++) {
    for (int c=1; c<length(monde,2)-1; c++) {
      int voisins = 0;
      for (int decLig=-1; decLig<=1; decLig++) {
        for (int decCol=-1; decCol<=1; decCol++) {
          if ([redacted]) {
            voisins = voisins + 1;
          }
        }
      }
      if ([redacted]) {
        mondeN[l][c] = true;
      } else if [redacted] {
        mondeN[l][c] = true;
      } else {
        mondeN[l][c] = false;
      }
      // Copie de mondeN dans monde
    }
  }
  continuer = readString();
}
```

Création de l'environnement

Décompte des voisines

Règles d'évolution

Cuisine interne ...

```
boolean[][] monde, mondeN;
// allocation de monde/N et
// initialisation de monde
```

Création de L'environnement

- Deux environnements à gérer : `monde` **et** `mondeN`
- Quels sont les paramétrages possibles ?
  - Pour la création ?
    - La taille de notre environnement (lig/col)
  - Pour l'initialisation ?
    - Aucune cellule vivante (pour `mondeN`)
    - Population aléatoire paramétrée par un pourcentage de cellules vivantes (?)
- Type de retour = tableau de booléens à 2D

```
boolean[][] monde, mondeN;
// allocation de monde/N et
// initialisation de monde
```

Création de L'environnement

1. réflexion sur l'usage et l'appel de la fonction

```
boolean[][] monde  = creer(30, 20, 0.3);
boolean[][] mondeN = creer(30, 20, 0.0);
```

2. réflexion sur le code fonctionnel, ie. le corps de la fonction, maintenant que la signature est connue

```
boolean[][] creer(int lig, int col, double tauxVivantes) {
  boolean[][] m = new boolean[lig][col];
  for (int l = 0; l < lig; l++) {
    for (int c = 0; c < col; c++) {
      m[l][c] = (random() < tauxVivantes);
    }
  }
  return m;
}
```

Simplification d'écriture au lieu d'utiliser une alternative

```
int voisins = 0;
  for (int decLig=-1; decLig<=1; decLig++) {
    for (int decCol=-1; decCol<=1; decCol++) {
      if [redacted] &&
                                          {
      voisins = voisins + 1;
      }
    }
  }
```

Décompte des voisines

- Quelles informations nécessaires pour faire le décompte des cellules voisines vivantes ?
  - correspond aux informations en entrée, ie. les paramètres de la fonction
- Quel est le résultat de la fonction ?
  - le nombre de cellules vivantes => un entier
- Travail sur la signature et seulement ensuite le corps

```
int voisins = 0;
  for (int decLig=-1; decLig<=1; decLig++) {
    for (int decCol=-1; decCol<=1; decCol++) {
      if (m[                                    ]
          (                                  )) {
      voisins = voisins + 1;
      }
    }
  }
```

Décompte des voisines

```
  int vivantes = compter(monde, l, c);
```

```
int compter(boolean[][] m, int lig, int col) {
  int voisinesVivantes = 0;
  for (int decLig=-1; decLig<=1; decLig++) {
    for (int decCol=-1; decCol<=1; decCol++) {
      if (m[                                    ]
          (                                  ) {
        voisinesVivantes = voisinesVivantes + 1;
      }
    }
  }
  return voisinesVivantes;
}
```

Signatures des fonctions

```
boolean[][] creer(int lig, int col, double taux)

void afficher(boolean[][] monde)

int compter(boolean[][] monde, int l, int c)

boolean evolution(int nbVoisins, boolean etat)

boolean[][] copier(boolean[][] source) OU
void copier(boolean[][] source, boolean[][]
cible)
```

Version décomposée

```
void algorithm() {
  boolean[][] monde  = creer(25, 40, 0.33);
  boolean[][] mondeN = creer(25, 40, 0.0);
  String continuer = "";
  while (!equals(continuer, "q")){
    for (int l=1; l<length(monde,1)-1; l++) {
      for (int c=1; c<length(monde,2)-1; c++) {
        int voisines = nombreDeVoisins(monde, l, c);
        mondeN[l][c] = evolution(voisines, monde[l][c]);
      }
    }
    monde = copier(mondeN);
    println(monde);
    continuer = readString();
  }
}
```

Extensions possibles

- Un monde torrique ?
  - En testant manuellement … bof
  - En recopiant l'information nécessaire !
- Une SDD plus efficace ?
  - Stocker les coordonnées des cases vivantes ?
- Une visualisation plus "jolie" ?
  - Des couleurs en fonction du nombre de voisins d'une cellule :)

Synthèse

- La notion de fonction va nous permettre de décomposer, factoriser notre code et améliorer la lisibilité
- Une fonction = un traitement
- Séparer les calculs des affichages (entrées/sorties)
- Toujours débuter par l'appel à la fonction
  - Identification des informations nécessaires
  - Définition de la nature du résultat
- Ensuite, s'attaquer au corps de la fonction

Analyse et modélisation

- Identifier les informations et les caractériser sous forme de données
- Identifier les structures de données possibles
- Analyser les principaux traitements à réaliser
  - Identifier celui/ceux semblant les plus critiques pour évaluer la SDD préférable
- Écrire une ébauche en pseudo-code des principales étapes de l'algorithme
- Affiner cette description avec du pseudo-code intégrant des appels à des fonctions (à créer plus tard) pour affiner la première ébauche
  - Être attentif aux données nécessaires pour que le traitement puisse être réalisé par la fonction !
- Passer seulement ensuite à l'implémentation sur machine
  - Ajouter les fonctions de tests au fur et à mesure de la création des fonctions
  - Tester le code après l'écriture de chaque couple fonction/fonction de test

John Conway talks about the Game of Life

Part 1 & Part 2 (bonus)