TP 7 et 9 : projet

1. Le jeu de la vie

Le jeu de la vie est un automate cellulaire dont les règles ont été définies par J. Conway en 1970. Le jeu se déroule sur une grille à deux dimensions, théoriquement infinie (mais de longueur et de largeur finies et plus ou moins grandes dans la pratique), dont les cases, qu'on appelle des «cellules» par analogie avec les cellules vivantes, peuvent prendre deux états distincts : «vivantes» ou «mortes».

L'état du jeu à l'étape n est uniquement fonction de son état à l'étape n-1. À chaque étape, l'évolution d'une cellule est entièrement déterminée par l'état de ses huit voisines de la façon suivante : (1) une cellule morte possédant exactement trois voisines vivantes devient vivante (elle naît), (2) une cellule vivante possédant deux ou trois voisines vivantes le reste, sinon elle meurt.

On peut également formuler cette évolution ainsi : (1) si une cellule a exactement trois voisines vivantes, elle est vivante à l'étape suivante, (2) si une cellule a exactement deux voisines vivantes, elle reste dans son état actuel à l'étape suivante, (3) si une cellule a strictement moins de deux ou strictement plus de trois voisines vivantes, elle est morte à l'étape suivante.

L'objectif de ce projet est de programmer une simulation du jeu de la vie sur plusieurs pas de temps, à partir d'une grille initiale.

2. Quelques variables et fonctions à définir pour commencer

Initialisation du jeu de la vie

1. Les variables globales `nb-line` et `nb-col` vont représenter respectivement le nombre de lignes et le nombre de colonnes de notre automate cellulaire. Définissez-les de façon à représenter par la suite un automate de 4 lignes et de 6 colonnes.

2. L'automate est représenté par une liste de listes comprenant des symboles : les cellules vivantes sont représentées par un * et les cellules mortes par un +. Nous appellerons `jeu-tableau` cette liste. Définissez-la pour la grille initiale suivante :

*+++++

**++*+

+**+*+

+++**+

Fonctions permettant l'affichage du jeu

Nous allons désormais créer des fonctions qui permettront d'afficher de manière lisible notre automate cellulaire.

3. Écrivez la fonction `displayLine` qui affiche une ligne de l'automate cellulaire.

```
(displayLine '(+ * * + * +)) → +*****+
```

4. Écrivez la fonction `displayBoard` qui affiche les différentes lignes de l'automate cellulaire.

```
(displayBoard jeu-tableau) →   *++++++
                               *+++++*
                               +*****+
                               ++++++*
```

N'hésitez pas ensuite à utiliser ces fonctions pour rendre plus lisible l'affichage de l'état de l'automate cellulaire.

Quelques fonctions utiles

5. Écrivez la fonction `ieme` qui renvoie le $i^{ième}$ élément d'une liste quelconque.

```
(ieme 3 '(a b c d e)) → c
```

Testez-la pour renvoyer par exemple la $2^{ième}$ ligne de l'automate cellulaire.

Les différentes cellules de l'automate cellulaire sont définies par un indice de ligne i et un indice de colonne j (la numérotation de ces indices commençant à 1).

6. Écrivez la fonction `alive?` qui renvoie la valeur 0 si la cellule de l'automate définie par les indices i et j est morte, et qui renvoie 1 si cette cellule est vivante. Cette fonction doit notamment vérifier la validité des indices i et j. Ainsi, toute cellule qui se trouve en dehors de l'automate est considérée comme morte.

```
(alive? 1 1 jeu-tableau) → 1
(alive? 0 1 jeu-tableau) → 0
```

Fonctions sur le voisinage des cellules

7. Écrivez la fonction `nbvoisins` qui renvoie le nombre de cellules voisines vivantes d'une cellule donnée de l'automate.

```
(nbvoisins 1 1 jeu-tableau) → 2
```

8. Écrivez la fonction `voisinageL` qui renvoie le voisinage, c'est-à-dire le nombre de cellules voisines vivantes, des différentes cellules de la ligne i de l'automate.

```
(voisinageL 1 jeu-tableau) → (2 3 1 1 1 1)
```

9. Écrivez la fonction `voisinage` qui renvoie le voisinage des différentes lignes de l'automate cellulaire.

```
(voisinage jeu-tableau) →
((2 3 1 1 1 1) (3 4 3 3 1 2) (3 3 3 5 3 3) (1 2 3 3 2 2))
```

Fonctions de mise à jour des états des cellules de l'automate

10. Écrivez la fonction `nextStateL` qui renvoie le changement d'états des cellules de la ligne i de l'automate cellulaire. On notera par un B une cellule qui doit renaitre à l'étape suivante et par un D une cellule qui va mourir.

```
(nextStateL 1 jeu-tableau) → (* B + + + +)
```

11. Écrivez la fonction `nextState` qui renvoie le changement d'états des différentes lignes de l'automate cellulaire.

```
(nextState jeu-tableau) →
((* B + + + +) (* D B B D +) (B * * + * B) (+ + B * * +))
```

12. Écrivez la fonction `updateCase` qui met à jour la valeur d'une cellule en fonction de son changement d'état : la cellule devient vivante si elle vient de renaître, et morte si elle vient de mourir. Si la cellule n'a pas connu de changement d'état, sa valeur reste inchangée.

```
(updateCase '*) → *          (updateCase '+) → +
(updateCase 'B) → *          (updateCase 'D) → +
```

13. Écrivez la fonction `updateLine` qui met à jour l'ensemble des cellules d'une ligne en fonction de leur changement d'état.

```
(define jeu-tableau-step1 (nextState jeu-tableau))
(updateLine (ieme 1 jeu-tableau-step1)) → (* * + + + +)
```

14. Écrivez la fonction `updateBoard` qui met à jour l'ensemble des lignes de l'automate en fonction du changement d'état des différentes cellules de l'automate.

```
(updateBoard jeu-tableau-step1) →
((* * + + + +) (* + * * + +) (* * * + * *) (+ + * * * +))
```

3. Le jeu de la vie sur plusieurs pas de temps

15. Écrivez la fonction `Play` qui permet d'effectuer la simulation du jeu de la vie sur plusieurs pas de temps. Cette fonction devra notamment s'arrêter à partir du moment où l'automate n'évolue plus dans le temps. Pour chaque pas de temps cette fonction devra afficher :

- Le pas de temps actuel
- L'état actuel de l'automate cellulaire
- Le voisinage des différentes cellules de l'automate
- Le changement d'état des cellules de l'automate

Vous testerez cette fonction sur l'automate cellulaire suivant :

```
*+++++
**+++*
+***+*
++++**
```

4. A la recherche de motifs intéressants

Il existe différentes structures intéressantes du jeu de la vie que vous pouvez tester :

- **Les structures stables** sont des ensembles de cellules ayant stoppé toute évolution : elles sont dans un état stationnaire et n'évoluent plus tant qu'aucun élément perturbateur n'apparaît dans leur voisinage. Un **bloc de quatre cellules** est la plus petite structure stable possible.
- **Les oscillateurs** se transforment de manière cyclique, en revêtant plusieurs formes différentes avant de retrouver leur état initial. Le **clignotant** est le plus petit oscillateur possible. Il est composé d'une cellule centrale et de deux cellules situées de part et d'autre ; si, sur une génération, ces cellules sont disposées horizontalement, elles apparaissent verticalement à la génération suivante donnant l'impression que la structure clignote.
- Il en existe bien d'autres que vous pourrez retrouver sur le Web.

5. Évaluation du projet

Lors de la séance d'évaluation du TP (**lundi 15 ou mardi 16 décembre**), vous devrez faire une démonstration de votre programme à votre enseignant/e. Il/elle vous demandera notamment d'expliquer les fonctions que vous avez implantées. Vous lui enverrez également par mail le **code source qui doit être commenté**, et qui doit comporter des **appels pertinents aux différentes fonctions** afin de les tester, ainsi que les résultats des tests que vous avez effectués.