Jeu de la Vie

Rappel : il est strictement INTERDIT d'utiliser l'opérateur `@` ainsi que le module `List`.

"Le jeu de la vie, automate cellulaire imaginé par John Horton Conway en 1970, est probablement, à l'heure actuelle, le plus connu de tous les automates cellulaires. Malgré des règles très simples, le jeu de la vie permet le développement de motifs extrêmement complexes."

– wikipedia

Les règles du jeu sont extrêmement simples, nous disposons d'un espace en deux dimension (théoriquement infini [1]) où chaque case contient une cellule qui peut être vivante ou morte. Toutes les cellules (vivantes ou mortes) interagissent avec ses huit cellules voisines. A chaque itération (cycle de vie) on applique les règles suivantes :

1. une cellule vivante qui a moins de deux cellules voisines vivantes meurt
2. une cellule vivante qui a deux ou trois cellules voisines vivantes survie au cycle
3. une cellule vivante qui a plus de trois cellules voisines vivantes meurt
4. une cellule morte qui a trois cellules voisines vivantes redevient vivante

Le but de ce TP va être d'implémenter votre version du jeu de la vie en `Caml` et cela en mode graphique. Pour stocker nos cellules nous utiliserons bien sur des listes, dans lesquelles chaque cellule sera représentée par un entier `0` pour les cellules mortes et `1` ou plus pour les cellules vivantes.

1. mais fini pour les besoins du TP

Palier 0 - Générateurs

0.1 Liste

Écrire `gen_list` qui génère une liste de taille `n` contenant uniquement des `0`.

```
# gen_list 10 ;;
- : int list = [0; 0; 0; 0; 0; 0; 0; 0; 0; 0]
```

0.2 Liste aléatoire

Écrire `gen_rand_list` qui génère une liste de taille `n` contenant aléatoirement [2] des `0` ou `1`.

```
# gen_rand_list 10 ;;
- : int list = [0; 1; 0; 1; 0; 0; 0; 0; 1; 1]
```

0.3 Liste de Listes

Écrire `gen_board` qui génère un "plateau carré" de coté `n` contenant uniquement des `0`.

```
# gen_board 3 ;;
- : int list list = [[0; 0; 0]; [0; 0; 0]; [0; 0; 0]]
```

0.4 Liste de Listes aléatoire

Écrire `gen_rand_board` qui génère un "plateau carré" de taille `n` contenant aléatoirement des `0` ou `1`.

```
# gen_rand_board 3 ;;
- : int list list = [[0; 1; 0]; [1; 1; 0]; [0; 0; 1]]
```

Palier 1 - Outils

1.1 Compte

Écrire `cell_count` qui retourne le nombre de cellules vivantes dans la liste passée en paramètre.

```
# cell_count [0; 1; 0; 1; 0; 0; 0; 0; 1; 1] ;;
- : int = 4
```

1.2 Re-Compte

Écrire `remaining` qui retourne le nombre de cellules vivantes de notre plateau passé en paramètre.

```
# remaining [[0; 1; 0]; [1; 1; 0]; [0; 0; 1]] ;;
- : int = 4
```

2. utiliser `Random.int`

Palier 2 - Affichage

Dans cette section vous allez être libre de choisir une couleur [3] pour les cellules vivantes, et une autre pour les mortes. Il va de soi que vous avez, et devez utiliser les fonctions de cet exercice ensembles et ne pas tout recoder à chaque fois.

2.1 Carré

Écrire `draw_square (x,y) size` qui affiche un carré noir de taille `size` à la position `(x,y)`.

```
val draw_square : int * int -> int -> unit = <fun>
```

2.2 Carré - encore

Écrire `draw_fill_square (x,y) size color` qui affiche un carré plein `color` de taille `size` en `(x,y)`.

```
val draw_fill_square : int * int -> int -> Graphics.color -> unit = <fun>
```

2.3 Cellule

Écrire `draw_cell (x,y) size cell` qui affiche une cellule (vivante ou morte) de taille `size` en `(x,y)`.

```
val draw_cell : int * int -> int -> int -> unit = <fun>
```

2.4 Plateau

Écrire `draw_board board size` qui affiche le plateau de taille de cases `size`.

```
val draw_board : int list list -> int -> unit = <fun>
```

Palier 3 - Accesseurs

Pour cette section, les fonctions retournerons une exception (via `failwith`) en cas d'impossibilité.

3.1 Cellule

Écrire `get_cell (x,y) board` qui retourne la valeur de la case demandée du plateau si possible.

```
# get_cell (2,1) ([[0;1;2];[3;4;5];[6;7;8]]) ;;
- : int = 3
```

3.2 Remplacement

Écrire `replace_cell value (x,y) board` qui remplace la valeur de la case `(x,y)` si possible.

```
# replace_cell 9 (2,1) ([[0;1;2];[3;4;5];[6;7;8]]) ;;
- : int list list = [[0; 1; 2]; [9; 4; 5]; [6; 7; 8]]
```

3.3 Donneur de vie

Écrire `seed_life board n` va donner vie à `n` cellules sur notre plateau.

```
# seed_life [[0; 0; 0; 0]; [0; 0; 0; 0]; [0; 0; 0; 0]; [0; 0; 0; 0]] 3 ;;
- : int list list = [[0; 1; 0; 0]; [0; 0; 1; 0]; [1; 0; 0; 0]; [0; 0; 0; 0]]
```

3.4 Voisinage

Écrire `get_cell_neighborhood (x,y) board` qui donne la liste cellules voisines [4] de celle en `(x,y)`.

```
# get_cell_neighborhood (2,1) [[11;12;13;14];[21;22;23;24];[31;32;33;34];[41;42;43;44]];;
- : int list = [11; 12; 22; 31; 32]
```

3. ou plus si affinité
4. huit au maximum : verticales, horizontales et diagonales

Palier 4 - Jeu

4.1 Itération

Écrire `iterate` qui donne le plateau de jeu après un cycle d'application de règles [5].

```
# iterate [[1; 0; 0; 0]; [0; 1; 1; 0]; [0; 0; 0; 0]; [0; 0; 1; 0]];;
- : int list list = [[0; 1; 0; 0]; [0; 1; 0; 0]; [0; 1; 1; 0]; [0; 0; 0; 0]]
```

4.2 Jouer

Écrire `play` qui va afficher le plateau, appliquer le cycle et recommencer tant qu'il y a de la vie [6].

```
val play : int list list -> unit = <fun>
```

Palier 5 - Bonii

5.1 Points de Vie

Nous allons considérer qu'une nouvelle cellule démarre sa vie avec dix points de vie.

5.1.1 Règles

Nous allons modifier les règles du jeu de la façon suivante :

1. une cellule vivante qui a moins de deux cellules voisines vivantes perd deux points de vie
2. une cellule vivante qui a deux voisines vivantes survie au cycle
3. une cellule vivante qui a trois voisines vivantes gagne un point de vie
4. une cellule vivante qui a plus de trois cellules voisines vivantes perd cinq points de vie
5. une cellule morte qui a trois cellules voisines vivantes redevient vivante avec dix points de vie

5.1.2 Adaptation

Vous allez devoir re-implémenter les fonctions afin de gérer des cellules ayant un nombre de points de vie variables. Il va de soi que l'affichage de la cellule (sa couleur) va dépendre directement de ses points de vie.

5.2 Créateur

Vous allez devoir gérer la création de cellules avec le clic de la souris.

5.3 Changement de dimension

Vous allez devoir implémenter le jeu de la vie sur un plateau à trois dimensions.

5.4 Et bien plus encore

Si vous n'en avez toujours pas assez, ou que ceux proposés ne vous conviennent pas, libre à vous d'en implémenter d'autres. Ceux-ci devront être explicitement décrits et commentés correctement.

5. relire l'introduction
6. *il y a de l'espoir* – Junior