game-of-life

February 5, 2018

Le but de ce projet est de programmer en Go le "jeu de la vie".
Le jeu de la vie fait évoluer une population de cellules sur une grille suivant les règles suivantes :

- une cellule apparait si elle a 3 voisines
- une cellule meurt si elle a moins de 2 ou plus de 3 voisines

Vous pourrez regarder chez vous la video suivante qui décrit bien ce jeu :

`https://www.youtube.com/watch?v=S-W0NX97DB0`

Cette vidéo décrit notamment certaines configurations particulières que nous chercherons à reproduire.
Avant de commencer l'implémentation, copiez et de-tarez dans votre $GOPATH/src le fichier suivant :

`Public/Busato/M1Info/ToBePutInSrcDir.tar.gz`

Dans le repertoire $GOPATH/src/game-of-life/ vous trouverez deux fichiers qui vont vous aider à faire l'implémentation :

- main.go : contient la fonction `main` (à ne pas toucher) et un objet de type `Grid`. Cet objet de type `Grid` est utilisé dans la fonction `GridGraph` qui elle même est appelée dans la fonction `main`. Il n'est pas implémenté, c'est à vous de le faire. Pour le faire, nous allons suivre les étapes décrites ci-dessous.
- plot.go : contient le code qui permet la représentation graphique (notamment la fonction `GridGraph`). Vous pouvez ne pas regarder ce fichier dans un premier temps et commencer l'implémentation directement, sans le comprendre (on le regardera à la fin si on a un peu de temps).

Il vous faut donc implémenter l'objet de type `Grid` dans `main.go`. Avant cela, nous allons créer un objet qui décrit la cellule.

- Création de la cellule:
  1. Créer une `struct` nommée `Cell`.
  2. Une cellule est décrite par sa position dans la grille et son état (morte ou vivante). Ajouter donc à `Cell` trois champs : le numéro de ligne dans la grille, le numéro de colonne dans la grille et une variable qui décrit son état.
  3. Lors de l'evolution, il faudra faire changer certaines cellule d'état (morte -> vivante ou vivante -> morte). Ajouter une méthode qui change l'état de la cellule.
- Création de la grille :
  1. Créer une `struct` nommée `Grid` contenant une `slice` de `slice` (donc une `slice` bi-dimensionnelle) de cellules. La `slice` de `slice` sera nommée `C` (c'est important car c'est ce dont a besoin plot.go pour afficher la grille).
  2. Créer une fonction `NewGrid()` qui retourne un pointeur vers un objet de type `Grid`. Dans cette fonction, on initialisera toutes les cellules dans l'état morte.

3. Créer une méthode `InitRandom()` qui initialise la grille de manière aléatoire. Écrire cette méthode de telle sorte à ce que chaque cellule soit dans l'état vivante avec une probabilité de 50%. Pour cela, vous pouvez faire des tirages aléatoires en utilisant le package `math/rand`.
4. Créer une méthode `Evolve()`. Pour l'instant, nous allons laisser cette méthode vide. Cela va nous permettre de tester le code que vous avez écrit jusqu'à présent.
5. Une fois que les étapes précédentes sont réalisées, compiler le programme avec `go build` et lancer l'exécutable. Vous devriez voir s'afficher quelque chose.
6. Dans la fonction `Evolve`, changer l'état d'une cellule quelconque. Relancer le programme. Vous devriez voir la cellule clignoter à chaque fois que vous appuyez sur la barre espace.

Si vous voyez la cellule clignoter après tous ces développements, bravo ! Vous avez fait la moitié du chemin. Vous pouvez maintenant poursuivre le programme en implémentant les règles de la vie.

- Implémentation des règles de la vie :

1. Pour faire évoluer la grille, il faut, pour chaque cellule, déterminer ses voisines. Ajouter une méthode `Neighbours` à la struct `Grid` qui prend une cellule et retourne une slice qui pointe vers ses voisines. La signature de cette méthode est donc :
   ```
   func (g *Grid) Neighbours(c *Cell) []*Cell
   ```
2. Executer cette méthode pour quelques cellules afin de vérifier qu'elle fait bien ce que vous attendez d'elle.
3. Implémenter les règles de la vie dans la méthode `Evolve` de `Grid` en utilisant la méthode `Neighbours` ci-dessus ainsi que la méthode de `Cell` qui change l'état des cellules.
4. Executer votre programme. Vous devriez observer une grille qui évolue.

Maintenant que vous avez un programme qui fait quelque chose, il faut le debugger ! Pour cela, nous allons non plus initialiser la grille de manière aléatoire mais de manière à avoir comme situation de départ les configurations simples décrites dans la vidéo mentionnée au tout début.

- Debuggage :

1. Initialiser la grille avec toutes les cellules mortes sauf trois cellules en ligne. Voyez vous le clignotant ?
2. Initialiser la grille avec toutes les cellules mortes sauf quatre cellules en ligne. Voyez vous la ruche ?
3. Initialiser la grille avec toutes les cellules mortes sauf cinq cellules en ligne. Voyez vous les quatre clignotants ?
4. Initialiser la grille avec une ruche plus une cellule vivante dans un coin de la ruche. Voyez vous les quatre ruches ?
5. Initialiser la grille avec cinq cellules vivantes : trois en ligne verticale, une au milieu à gauche de la ligne et une en haut à droite. Voyez vous quelques chose de semblable à ce qui est montré dans la vidéo ? Notamment, voyez vous des planeurs ?
6. Arrivez-vous à faire le canon à planeur ?

Si votre programme passe ses tests de debuggage avec succès alors vous pouvez considérer que votre programme marche. Félicitations !

Vous pouvez maintenant vous amuser en testant d'autres configurations. Have fun !

In [ ]: