Consignes communes aux DM/Projets :

- il vous est demandé d'écrire/compléter un code Python : il est impératif d'écrire de la documentation et des commentaires dans ces codes ;
- il est préférable de rendre un code pas tout-à-fait fonctionnel, mais de vous, plutôt qu'un code recopié ici ou là, cela se voit toujours !
- si un document vous est demandé en plus, vous pouvez le rendre, au choix, en version papier ou électronique (dans ce cas, à rendre dans moodle avec votre fichier `.py`) ;
- n'oubliez pas que vous devez indiquer vos noms/prénoms/classes/prof dans *tous* les documents que vous rendez (papier ou fichiers) !

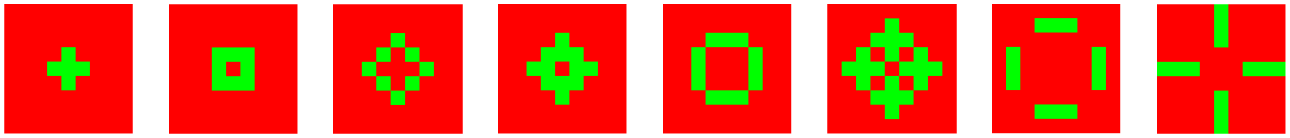
Le *jeu de la vie* de CONWAY est un exemple d'automate cellulaire[1]. Le principe est d'étudier l'évolution de l'état de cellules dans une grille. Dans la version de CONWAY, on ne s'intéresse qu'aux huit voisins de chaque cellule dans un plan pour décider de son état futur (deux états possibles) mais d'autres versions permettent de modéliser des systèmes dans lesquels on regarde plus loin autour de chaque cellule, avec une modélisation 3D, ou avec plus de deux états possibles...

On considère un quadrillage dans lequel chaque case est une cellule qui peut être, au choix[2], dans deux états : vivante ou morte. L'objectif est d'étudier au cours du temps l'évolution de l'état des cellules de la grille. À chaque intervalle de temps, il faut compter le nombre de cellules mortes et vivantes parmi les huit cellules qui entourent chacune d'elles[3]. À l'étape suivante, on déterminera l'état de chaque cellule en respectant les deux règles ci-dessous :

- si une cellule morte a exactement trois cellules vivantes autour d'elle, elle devient vivante à l'étape suivante, sinon elle reste morte ;
- si une cellule vivante est entourée de deux ou trois autres cellules vivantes, elle reste vivante, sinon, elle meurt.

L'objectif de ce devoir est d'écrire certaines fonctions permettant de simuler l'évolution d'un tel système. La situation de départ sera définie sous la forme d'un fichier texte (des exemples sont fournis sur moodle), vous aurez à compléter le code (fourni également) dans lequel certaines fonctions sont déjà écrites.

Exemple d'évolution :



Puis, les deux dernières étapes se reproduisent indéfiniment...

Les questions 1 à 3 sont à faire sur feuille à rendre (ou dans un document pdf à remettre avec le fichier `.py` sur moodle).

1. Justifier qu'un automate de base constitué de quatre cellules vivantes disposées en carré est *stable*[4] (on pourra par exemple reproduire la figure dans un quadrillage en indiquant le nombre de voisins vivants dans chaque case).

1. Un *automate cellulaire* permet de modéliser des phénomènes physiques dynamiques.
2. Choix exclusif !
3. Huit au plus car les cellules « au bord de la grille en ont moins.
4. Un automate est dit stable s'il n'évolue pas au cours du temps.

2. Proposer un autre automate stable. Justifier qu'il est stable.
3. Proposer un automate (différent de la situation rencontrée dans l'exemple ci-dessus) périodique[5]. Justifier par une méthode analogue à la question 1.
4. Compléter le code fourni sur moodle en respectant les consignes données ci-après.
   a. Écrire la fonction `affiche` qui affiche au format texte l'état de l'automate. Elle prend en paramètre une liste de listes composées de `0` et de `1`.
   Vous pourrez tester votre fonction en écrivant par exemple :
   ```
   a = lecture('automate1')
   affiche(a)
   ```
   Attention, dans Pyzo, il faudra exécuter le fichier avec les touches `ctrl-maj-E`.
   b. Compléter la fonction `compte_voisins` documentée dans le code fourni. Il peut être utile aussi de compléter la fonction `affiche_nb_voisins` pour vérifier votre fonction `compte_voisins` sur des exemples simples.
   c. Compléter la fonction `calcul_etape` (la documentation est fournie).
   À partir de maintenant, vous pouvez tester votre *jeu de la vie* grâce à la fonction `simul_graphique` fournie. Par exemple :
   ```
   simul_graphique('automate1.txt')
   simul_graphique('automate2.txt')
   ...
   ```
   Si vous avez un message d'erreur dans la console vous indiquant que Python ne connait pas la bibliothèque `pygame`[6] il suffit de l'installer avec la commande suivante à écrire dans la console :
   ```
   pip install pygame
   ```
   d. Dernière étape obligatoire : compléter la fonction `simul_txt`. Cette fonction doit :
   — simuler le jeu de la vie en mode texte : affichage dans la console de chaque étape ;
   — écrire dans des fichiers les différentes étapes sous la forme de lignes de `0` et de `1` en s'arrêtant au nombre d'étapes éventuellement passé en paramètre. Il faudra pour cela utiliser la fonction `ecriture` fournie et documentée.
   e. Compléter la fonction `crea_map_alea` (documentée) permettant de créer un automate de départ aléatoire et de l'écrire dans un fichier.
   f. (Bonus pour ceux qui connaissent déjà pygame) Écrire un programme Python avec Pygame permettant de créer un fichier de départ en cliquant sur un quadrillage.

5. On dit d'un automate qu'il est périodique s'il se reproduit identique à lui même après n étapes ($n > 0$).
6. Nous apprendrons à utiliser cette bibliothèque un peu plus tard dans l'année.