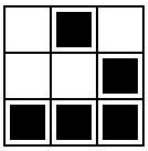
PROJET : LE JEU DE LA VIE

Stéphane GAREDDU



THE GAME OF LIFE

TERMINALE NSI - LYCÉE FESCH AIACCIU

Table des matières

A) Présentation du dossier

Ce dossier résume est un résumé dont la finalité est d'implémenter une adaptation simplifiée du fameux ***jeu de la vie*** (game of life) au moyen du langage Python (>3.x). Ce « jeu » a été imaginé par le mathématicien britannique John Horton Conway, disparu l'an dernier. Formellement, le jeu de la vie décrit un automate cellulaire bidimensionnel, qui est également Turing-complet.

Nous avons décomposé le problème en sous-problèmes, répartis en activités. Ceci permet de comprendre, même avec des simplifications et des abstractions éventuelles, quelles sont les étapes de création du programme et comment les imbriquer. Il est également judicieux d'aborder ces activités **avec un crayon et une feuille**, plutôt que de se précipiter sur les machines pour écrire du code. La réflexion sur papier reste, en algorithmique comme dans d'autres disciplines de l'informatique, fondamentale pour bien représenter ce que l'on veut coder (croquis, pseudo-langage…) mais aussi de vérifier un algorithme avant de le coder sur machine, en le « faisant tourner » à la main.

L'approche privilégiée est, dans la quasi-intégralité de ce document, l'approche naïve : on détermine, pour toute la grille de jeu, le futur de chaque cellule, sans chercher à optimiser.

B) Prérequis et culture d'informatique fondamentale

Répondre en quelques lignes aux questions suivantes :

1. Qu'est-ce qu'un automate ?
2. Quelles sont des applications d'un automate ?
3. Qu'est-ce qu'une machine de Turing ?

4. Quelles sont les applications d'une machine de Turing ?
5. Qu'est-ce qu'un automate cellulaire ?
6. Quelles sont les applications d'un automate cellulaire ?

C) Activité 1 : Comprendre / appliquer manuellement les règles d'évolution du jeu de la vie

Consigne et questions : Le jeu de la vie (game of life) se joue sans joueurs. Ce jeu, plus exactement ce jeu de simulation, est composé d'une ***grille*** (théoriquement infinie) de ***cellules***. On considèrera que cette ***grille*** est carrée, de dimension finie et qu'elle possède des bords. Chaque ***cellule*** est soit dans l'état « ***morte*** » soit dans l'état « ***vivante*** ». Il n'y a pas d'autre état possible. À partir d'une situation initiale donnée, le jeu évolue automatiquement selon des ***règles*** simples, et au tour de jeu suivant apparaît une nouvelle configuration. On appelle chaque tour de jeu une ***génération***. Les règles d'évolution sont :

- Si une cellule morte a exactement 3 voisines vivantes, elle vit (naît) à la génération suivante
- Si une cellule vivante a 2 ou 3 voisines vivantes, elle reste vivante à la génération suivante
- Si une cellule vivante a moins de 2 voisines vivantes ou plus de 3 voisines vivantes, elle meurt

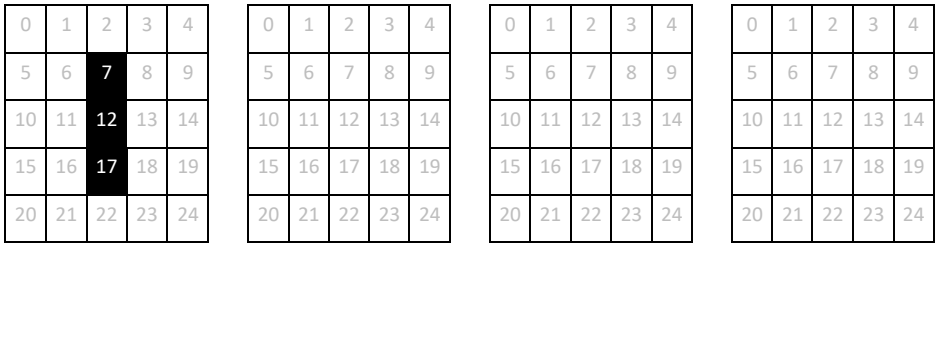


Figure 1

Q1 : À partir de la configuration initiale (génération 0) de la Figure 1, dessinez l'évolution pour générations suivantes
Q2 : Prolongez de quelques générations : que remarquez-vous ? Cela aurait-il changé quelque chose que les cellules initialement vivantes soient la 11, la 12 et la 13 ?
Q3 : Il a été montré que ce que vous obtenez était le plus petit oscillateur de ce jeu. Sa période est seulement de 2 générations. Chaque structure particulière de ce jeu porte un nom : quel est celui de cette structure ?
Q4 : Pourquoi est-il « dangereux » de travailler sur une seule et même grille ?
Q5 : (Culture informatique) Le moteur de recherche Google regorge d'« Easter eggs ». Qu'est-ce donc ? Pour le comprendre, tapez comme requête « jeu de la vie » sur ce moteur de recherche

D) Activité 2 : Représenter la grille (le plateau de jeu)

Consigne et questions :
Q1 : Proposez une structure de données Python adaptée pour représenter la grille de jeu (le plateau)

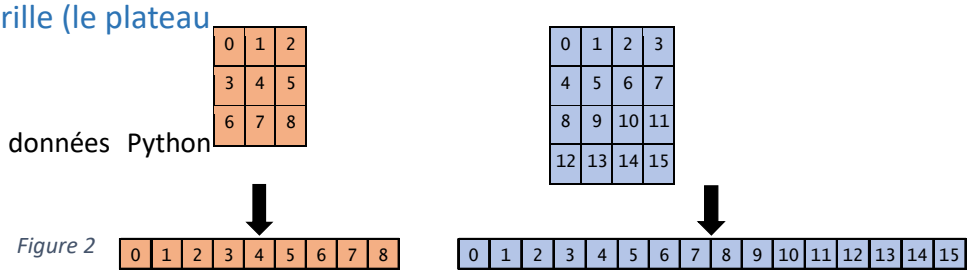


Figure 2

Notes : Il est intéressant de leur faire choisir une liste unidimensionnelle plutôt qu'à 2 dimensions : il n'est pas toujours opportun de « forcer » la machine à utiliser une représentation d'un problème proche de la nôtre. Cela offre des exercices intéressants, par exemple passer d'un tableau (vu en « 2d ») à une variable de type liste (unidimensionnelle) (voir Figure 2)

E) Activité 3 : Représenter une cellule vivante / morte

Consigne et questions :

Q1 : Proposez une manière de représenter le fait qu'une cellule peut être morte ou vivante

Q2 : Écrivez une fonction qui vérifie si une cellule (passée en paramètre par son indice `ic`) est morte ou vivante. On suppose que liste `l` symbolisant le plateau est aussi passée en paramètre. Expliquez son manque d'intérêt

F) Activité 4 : Représenter une configuration initiale grâce aux 2 points précédents

Consigne et questions : Soit une grille de jeu de 25 cellules comme le montre la Figure 1

Q1 : Soit une variable Python `cote` contenant un entier représentant le côté d'une grille de jeu, créez une fonction commentée `initGrille(cote)` qui crée une liste de cellules (attention à la taille) initialisées à « morte ».

Q2 : Écrivez la fonction `ajouteVivantes(liste)`. Cette fonction admet une liste d'indices de cellules vivantes, et agit sur une variable `grilleJeu` globale en mettant à `True` les cellules aux indices contenus dans `liste`

G) Activité 5 : Déterminer les voisines d'une cellule

Consigne et questions : On se propose de déterminer les voisines d'une cellule. Pour ce faire, on a besoin de distinguer tous les cas possibles

Q1 : Combien en existe-t-il et quels sont-ils ?

Q2 : (cas général) Soit une cellule d'indice `i` située hors des bords et des coins d'un plateau de jeu de côté `cote`, soient vD, vBD, vB, vBG, vG, vHG, vH, vHD les 8 voisines respectivement droite, bas-droite, bas, bas-gauche, gauche, hautgauche, haut, et haut droit de cette cellule. Déterminer les valeurs des indices de chaque voisine en fonction de `cote` et de `i`

Q3 : (cas particulier, bord) Écrivez le code des 4 fonctions suivantes : `estBD(i)`, `estBB(i)`, `estBG(i)`, `estBH(i)` qui prennent en paramètre l'indice `i` d'une cellule. Le côté de la grille, `cote`, est supposé connu (variable globale). Le but de ces fonctions est de déterminer respectivement si la cellule est sur le bord droit, bas, gauche ou haut plateau de jeu. Elles renverront `True` si la cellule est sur le bord spécifié, `False` sinon

Q4 : (cas particulier, coin) En déduire les fonctions `estCoinHD(i)`, `estCoinBD(i)`, `estCoinBG(i)`, `estCoinHG(i)` les fonctions qui renvoient `True` si la cellule coïncide avec, respectivement, le coin haut-droit, basdroit, bas-gauche, haut-gauche de la grille, et `False` sinon

Q5 : Créez à partir des deux questions précédentes deux fonctions auxiliaires : `estBord(i)`, `estCoin(i)`

Q6 : À partir de tout cela, écrire le code d'une fonction `voisines(i)` qui renvoie une liste des indices des cellules voisines de cette cellule en fonction de son indice `i`

Notes : Il est intéressant de comparer deux approches : l'une, comme indiqué, utilise le côté comme étant une variable globale (cela simplifie le code mais rend les fonctions interdépendantes – trop couplées), l'autre le passe en paramètre (code plus compliqué et lourd, mais réutilisable car générique).

H) Activité 6 : Compter les voisines vivantes d'une cellule

Consigne et questions : On veut trouver combien de voisines vivantes possède une cellule ***Q1 :***
Que doit-on absolument connaître pour pouvoir programmer une telle fonction ?

Q2 : Implémentez la fonction `voisinesVivantes(i)` qui renvoie un entier correspondant au nombre de cellules vivantes voisines de la cellule d'indice `i`

Q3 : Considérons la Figure 1, génération 0, que renvoient : `voisinesVivantes(0)`, `voisinesVivantes(23)`, `voisinesVivantes(18)`, `voisinesVivantes(11)` ?

I) Activité 7 : Déterminer les règles d'évolution pour une cellule, puis pour tout le jeu

Consigne et questions : À l'aide des résultats de l'activité précédente, on se propose d'implémenter une fonction `dewie(i)` qui prend en paramètre un entier `i` (indice d'une cellule) et renvoie le futur de la cellule : `False` pour

morte, `True` pour vivante. Par rapport aux règles initiales, on peut d'ailleurs apporter de petites simplifications. ***Q1 :*** Implémentez la fonction `dewie(i)`

Q2 : Implémentez, à l'aide de tout ce qui a été vu précédemment la fonction `nextGeneration(grille)` qui prend en paramètre une grille et renvoie une grille représentant l'état du jeu à la génération suivante.

Notes : On doit ici comprendre la nécessité de travailler sur deux grilles afin d'éviter les effets indésirables.

J) Activité 8 : Implémenter / utiliser un algorithme de simulation

Consigne et questions : On nous fournit une fonction `affichageBasique(grille)` qui affiche une grille de jeu dans une console système.

Q1 : Proposez une implémentation de la fonction `moteurJeu(coteU, listeVivantes, nbTours)` qui prend en paramètre un côté, une liste d'indices de cellules vivantes (config. initiale) et un nombre de générations

Q2 : Soit un plateau de jeu de côté ***cote*** représenté par une liste unidimensionnelle. Quelle est, en fonction de ***cote***, la complexité d'un algorithme qui parcourt chacune des cellules de ce tableau ? On décide maintenant d'utiliser une liste à deux dimensions pour représenter ce plateau. Écrivez l'algorithme qui parcourt chaque cellule de ce plateau. Quelle est, en fonction de ***cote***, la complexité d'un tel algorithme ?

Q3 : On se pose à présent la question suivante : est-il nécessaire de parcourir toutes les cellules de la grille à chaque génération pour calculer la configuration à la génération suivante ? Quelles sont, en réalité, les seules cellules dont l'état peut être modifié entre 2 générations ?

Q4 : Proposez une fonction « naïve » `vivantesEtVoisines(grille)` qui prend en argument une grille de jeu et renvoie uniquement les indices des cellules vivantes et leurs voisines. Quel est, au plus (dans le « pire des cas »), en fonction de n (nombre de cellules vivantes), le nombre maximal de cellules que renvoie cette fonction ?

Q5 : Testez votre fonction sur quelques cas concrets, par exemple une grille contenant un clignotant. Que remarquezvous ? Comment faire pour remédier à cela ? Modifiez la fonction précédente en conséquence

Q6 : (Bonus) Avec seulement 7 cellules actives dans la situation initiale, nous avons obtenu le « clown » de la page de garde au bout de 110 itérations. Faites de recherches pour retrouver ces cellules et lancez une simulation permettant de les obtenir (côté 50).

K) Références des ressources utilisées

- Article de blog de D. Louapre sur le Game of Life, https://scienceetonnante.com/2017/12/08/le-jeu-de-la-vie/
- Vidéo associée : https://www.youtube.com/watch?v=S-W0NX97DB0&ab_channel=ScienceEtonnante
- Page Wikipédia du jeu de la vie, https://fr.wikipedia.org/wiki/Jeu_de_la_vie
- Eduscol : livret d'algorithmique et programmation niveau lycée
- Learn Python 3 the Hard Way: A Very Simple Introduction to the Terrifyingly Beautiful World of Computers and Code, éditions Addison Wesley, Z. Shaw, ISBN 978-0-134-69288-3
- Article du Scientific American de 1970, https://web.stanford.edu/class/sts145/Library/life.pdf
- Une machine de Turing dans le Game of Life, P. Rendell, https://www.ics.uci.edu/~welling/teaching/271fall09/Turing-Machine-Life.pdf Une vidéo spectaculaire sur le game of life : https://www.youtube.com/watch?v=C2vgICfQawE&ab_channel=RationalAnimations
- L'explication concernant le choix de la fonction « God », celle qui détermine l'avenir d'une cellule, fonction que nous avons appelée « dewie » (du nom d'un personnage de la série TV Malcolm) https://www.youtube.com/watch?v=mfPYfegOPwk&ab_channel=30Toshirock