Sorbonne Université

PROJETS Matlab

Cours 2025-2026

Licence Professionnelle (L3)

Instrumentation Optique et Visualisation

Quentin Glorieux
quentin.glorieux@lkb.upmc.fr

1 Projet

1.1 Evaluation

L'UE de Matlab est évaluée sur le projet encadré que vous allez effectuer. Pour ce projet vous disposez de 4 séances de 4 heures avec un encadrant et d'un temps en autonomie de 3 semaines pour finaliser le code et préparer la présentation. Le projet doit être effectuer en binôme. La note pourra être différente entre les deux membres du binôme si il est noté une grande disparité dans l'implication et la connaissance du code.

La note se décompose de la façon suivante :
— 20% sur l'avancement dans le projet (jusqu'où avez vous pousser le projet ?)
— 10% sur la qualité du code (commentaires, organisation des fonctions...)
— 35% sur le rapport (présentation soignée, qualité de la rédaction...)
— 35% sur la soutenance orale

La difficulté des projets est variable. Certains sont plus guidés que d'autres. Selon votre niveau en Matlab, choisissez un projet adapté. Pour valoriser les projets les plus difficiles une coefficient $\times 1.2$ sera appliqué au projets $\star\star$ et un coefficient $\times 1.3$ aux projets $\star\star\star$. Ce coefficient s'applique sur la note d'avancement uniquement.

Attention : *Il existe des codes sur internet pour des projets similaires, nous le savons et nous vérifierons. Vous devez comprendre parfaitement votre code pour la soutenance.*

1.2 Liste des projets

— Projet 1 : **Codage et décodage du signal sonore.** . Difficulté : $\star$
L'objectif de ce projet est de décoder un signal sonore correspondant à un numéro de téléphone. Le sujet est très détaillé et guidé.

— Projet 2 : **Stéganographie.** . Difficulté : $\star$ (ou $\star\star$ si vous faites la partie c)
L'objectif de ce projet est d'apprendre à cacher un message de façon indetectable dans une image. Le sujet est très détaillé et guidé, il est cependant un peu plus long que le projet 1.

— Projet 3 : **Marche aléatoire.** . Difficulté : $\star\star$
L'objectif de ce projet est d'étudier la dynamique aléatoire d'une marcheur en une dimension. Ce comportement aléatoire simule relativement bien le mouvement Brownien (mouvement aléatoire d'une « grosse » particule immergée dans un fluide et qui n'est soumise à aucune autre interaction que des chocs avec les « petites » molécules du fluide environnant.) Le projet est bien guidé mais assez difficile au niveau programmation.

— Projet 4 : **La planche de Galton.** . Difficulté : $\star\star$
L'objectif de ce projet est d'étudier simuler une planche de Galton pour réaliser des expériences statistiques. Il est relativement simple à coder mais assez peu guidé et demandera donc une important capacité de modélisation du problème.

— Projet 5 : **Le jeu de la vie.** . Difficulté : $\star\star$ (ou $\star\star\star$ si vous faites les questions IV et V)
L'objectif de ce projet est d'étudier le jeu de la vie de John Conway. C'est un jeu avec des règles très simples qui a des comportements très surprenants, dit émergeants. Ce projet est plus difficile que les précédents mais très bien guidé pour vous permettre d'avancer.

— Projet 6 : **La bataille.** . Difficulté : $\star\star\star$
L'objectif de ce projet est de simuler une partie du jeu de carte bien connu. Ce projet est libre (pas de direction prédéfinie) et demande donc une bonne dose de modélisation du problème et des capacités en programmation. A ne choisir que si vous êtes très à l'aise.

1.3 Choix des projets

Une inscription par binome. Pas plus de 1 binome par projet.

Projet Matlab : Codage et décodage du signal sonore d'un numéro de téléphone.

Tom Darras - Quentin Glorieux

tom.darras@lkb.upmc.fr - quentin.glorieux@lkb.upmc.fr

Dans les téléphones modernes, les codes DTMF (dual-tone multi-frequency) sont utilisés pour la composition des numéros de téléphone. En pratique, chaque touche du téléphone est associée à un couple de fréquences audibles jouées simultanément. On peut ainsi coder 16 touches en utilisant 8 fréquences différentes. Ces fréquences peuvent être reconnues par des dispositifs pour déterminer quel numéro à été composé. Les fréquences des touches sont déterminées pas le tableau suivant :

	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

Le but de ce projet sera d'une part de créer le signal sonore associé à la composition d'un numéro de téléphone, et d'autre part de réaliser le décodage d'un numéro de téléphone inconnu.

I. CRÉATION DU SIGNAL SONORE CORRESPONDANT À UNE TOUCHE.

L'objectif de cette partie est de recréer les signaux sonores correspondants aux différentes touches du clavier téléphonique.

1. Créer une fonction Matlab 'note.m' permettant de générer un signal composé de deux fréquences *f1* et *f2* pendant une durée T, avec une fréquence d'échantillonnage *Freq*. De manière générale, comment doit-on choisir la fréquence d'échantillonnage ? Dans la suite on prendra comme valeurs $T = 100$ ms et $Freq = 10$ kHz.

2. Tracer les signaux correspondants aux différents symboles du clavier téléphonique en fonction du temps.

3. A l'aide de la commande *sound()*, tester les signaux sonores associés aux différents symboles.

4. Créer une fonction Matlab 'TFourier.m' permettant d'obtenir la transformée de Fourier d'un signal échantillonné à une fréquence *Freq*.

5. Tracer les spectres associés aux signaux sonores des différentes touches du clavier. Vérifier que cela correspond à ce qui est attendu.

6. Créer un script 'lecture_numero.m' prenant en entrée le numéro de téléphone d'un utilisateur et donnant en sortie le signal sonore correspondant. On n'oubliera pas d'ajuster le délai temporel entre les différentes touches.

II. DÉCODAGE D'UN NUMÉRO DE TÉLÉPHONE INCONNU.

L'objectif de cette partie est de retrouver un numéro de téléphone inconnu à partir du signal sonore émis par les touches du clavier lors de sa composition.

1. Charger le fichier 'numero_inconnu.wav' en utilisant la commande *audioread()*. Quelle est la fréquence d'échantillonnage du signal ? Quelle est la durée du signal ? Combien de touches ont été composées ?

2. Décomposer le signal en *N_notes* sous-signaux, où *N_notes* est le nombre de note du numéro de téléphone. Tracer ces signaux en fonction du temps.

3. Pour chaque sous-signal, réaliser sa décomposition spectrale en utilisant la transformée de Fourier. Associer à chaque sous-signal la touche du clavier correspondante. On pourra automatiser la méthode en utilisant un seuillage.

Matlab Projet : Stéganographie

1. Introduction

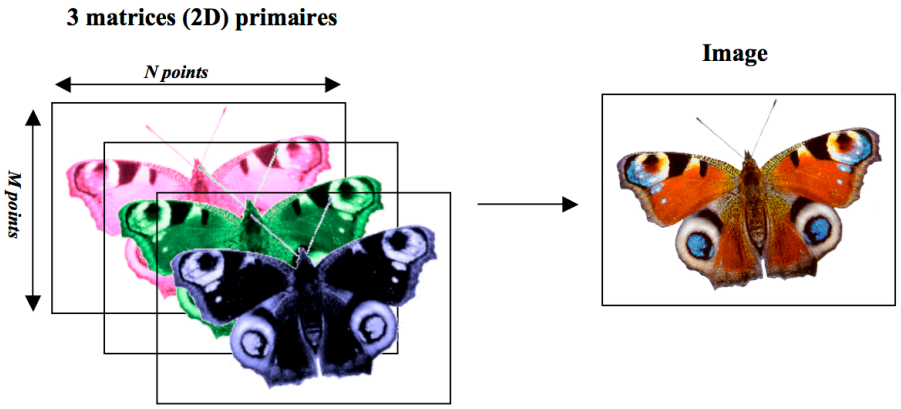
La stéganographie (du grec steganos : couvert et graphein : écriture) consiste à dissimuler une information au sein d'une autre à caractère anodin, de sorte que l'existence même du message caché passe inaperçue. Alors qu'avec la cryptographie, la sécurité repose sur le fait que le message ne sera sans doute pas compris, la stéganographie, s'appuie sur le fait que le message ne sera meme pas détecté.

Ce projet vous propose de mettre en application l'une des méthodes les plus basiques, en termes de stéganographie. L'idée est de prendre une image et de la modifier de manière imperceptible afin d'y dissimuler l'information à transmettre, en l'occurrence une autre image binaire (composée de deux niveaux : 0 et 1) contenant du texte.

2. Quelques notions essentielles

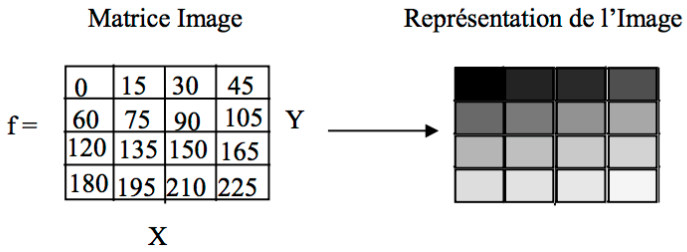
a. Codage des images sous matlab

Une image n'est autre que l'extension d'un signal à 2 dimensions. Une image couleur est la superposition de 3 composantes de base (Rouge, Vert et Bleu). Sous Matlab une telle image peut être codée par une matrice tridimensionnelle. Elle correspond à la mise en cascade des 3 matrices (2D : N*M) correspondant aux 3 composantes primaires (Rouge Vert et Bleu). Chacune de ces 3 matrices primaires (aussi appelées « plans couleur ») contient le niveau de couleur pour chaque point de l'image considérée. En général chaque niveau de couleur est codé entre 0 (canal colorimétrique éteint) et 255 (canal colorimétrique maximum) ce qui correspond à 2553 = 16 581 375 combinaisons de couleurs possibles.



Dans le cas particulier des images en noir et blanc, les 3 composantes de chaque point de l'image considérée (pixel) sont égales. Nous avons donc 256 niveaux de gris (de 0 pour le noir à 255 pour le blanc, en passant par un gris moyen pour 128).

Remarque : *Lorsque l'on charge une image sous Matlab, la matrice correspondante est de type uint8 (codage sur 8 bits non signés, autrement dit : 0-255). Pour simplifier les calculs et pouvoir utiliser toutes les fonctions matlab sur ces données, il est préférable de les convertir en réels (commande « double »), puis après traitement les reconvertir en entier codé sur 8 bits nonsigné avant de sauvegarder en format bmp (commande «uint8 »).*



b. <u>Une technique de stéganographie</u>

La technique de base, dite LSB (pour Least Significant Bit ou bit de poids faible, en français) consiste à modifier le(s) bit(s) de poids faible des pixels codant l'image. En effet, supposons que le niveau des couleurs soit codé sur 8 bits (256 niveaux, comme nous l'avons vu précédemment), le fait de modifier le ou les bits de poids faible entraînera un changement de couleur presque imperceptible pour le pixel considéré.

Cette technique de stéganographie très basique s'applique tout particulièrement au format d'image BMP, format sans compression destructive, avec codage des pixels sur 3 octets (3*8 bits) comme énoncé ci-dessus.

3. Travail à réaliser

Le logiciel que vous allez constituer, va donc permettre, à l'aide de la méthode de sténographie présentée ci dessus, de crypter une information cachée dans une image. Cette information sera sous la forme d'une image binaire (en noir et blanc) de texte, comme dans l'exemple ci-dessous :

L'orang-outan appartient à la famille des pongidés qui comporte deux sous-espèces : pongo pygmaeus pygmaeus qui vit à Bornéo, et pongo pygmaeus abelii que l'on trouve à Sumatra. Sa taille varie entre 110 cm pour la femelle et 140 cm pour le mâle, tandis que son poids oscille entre 30 et 90 kg selon le sexe. La gestation dure 245 jours, et il n'y a qu'une seule naissance tous les 4 ou 5 ans. Son espérance de vie atteint une quarantaine d'années. Peu fourni sauf sur les épaules où il peut atteindre 50 cm de long, son pelage est d'un brun plus ou moins roux. Il occupe exclusivement les forêts ombrophiles de Malaisie (provinces de Sabah et Sarawak), du Sultanat de Brunei Darrussalam, et d'Indonésie (Kalimantan et Sumatra). En malaisien, son nom signifie « homme de la forêt »

Dans cet exemple, le texte est codé dans le dernier bit du plan « rouge » de l'image de droite. Un bit à 1 correspondra à un pixel blanc et un bit à 0 un pixel noir de l'image de texte. Il suffit donc d'extraire la matrice (composée de 0 et 1) de l'image de ouran-outan pour retrouver le texte (sous la forme d'une image noir et blanc).

Afin de constituer ce logiciel, vous allez procéder par étapes :

• la première etape est le codage de méthodes simplifiées de cryptage et décryptage, où l'information cachée ne sera cryptée que dans un seul plan (à la manière de l'exemple de l'oran-outan) et où les images cachées seront crées à l'aide d'un logiciel externe à Matlab (ImageJ ou MS Paint).

• la seconde étape introduit la notion d'une clef de cryptage qui sera indispensable pour décrypter correctement l'image. Seules les personnes connaissant cette clef pourront décrypter simplement l'image.

• La troisième étape consistera à créer une interface graphique pour un utilisateur non-spécialiste de Matlab et ajouter la possibilité de créer une image de texte à partir de Matlab (sans passer par un logiciel externe).

Il sera important de créer une interface utilisateur transparente (c'est-à-dire que des personnes ne connaissant quasiment rien en Matlab puissent utiliser), en utilisant notamment les fonctions d'entrées-sorties (« input », « disp »,…) ainsi que des menus à boutons.

a. Etape 1 : Codage simplifié

Lors de cette étape, il ne s'agit pas encore de créer le logiciel final, mais simplement pour vous de mettre en pratique, et ainsi de maîtriser, les méthodes de cryptage et décryptage.

Tout d'abord, vous devez prendre une image quelconque (cherchez la sur internet par exemple) que vous devez convertir en format bmp (à l'aide de Matlab – avec « imread » et « imwrite »). Nommons-la « image.bmp » pour faciliter la suite des explications.

Ensuite à l'aide de ImageJ, créez une image toute blanche. Dans cette seconde image, à l'aide de ImageJ, écrivez quelque chose en noir. Sauvegardez en format bmp (monochrome) sous un autre nom (par exemple « imagetexte.bmp »).

Ensuite, créez deux scripts « cryptage.m » et « decryptage.m » :

• Le script cryptage.m doit charger les deux images, puis remplacer le bit de poids faible du plan « rouge » de l'image « image.bmp » avec l'information contenue dans l'image « imagedetexte.bmp ». Puis sauvegarder l'image obtenue sous format « bmp » (sous le nom « imagecrypte.bmp »).

• Le script decryptage.m doit charger l'image « imagecryptee.bmp » pour en extraire du plan « rouge » l'information cryptée.
*Nous vous rappelons que la valeur d'un pixel est codée sur 8 bits, selon la relation suivante : valeur = bit0 + bit1*21 + bit2*22 + bit3*23 + … + bit7*27 . Par exemple : 173 = 1+0*2+1*4+1*8+0*16+1*2^5+0*2^6+1*2^7 -> 10101101*

Vous pouvez utiliser la fonction imagesc de Matlab pour visualiser les différentes images.

b. Etape 2 : Clé de cryptage

Précédemment, vous avez codé l'information cachée dans un seul plan (le « rouge ») de l'image couleur. Cependant, toute personne au fait de la technique serait en mesure de décrypter votre image (et donc de lire l'information cachée).

A présent, vous allez introduire une clef de cryptage : seule la personne qui connaît aussi la clef sera en mesure de décrypter votre image. Dans cette partie, la clef consistera en une suite de 1, 2 et 3, correspondant aux plans couleurs où sera codée l'information : ainsi si la clef est 231321, le premier pixel de l'image « imagedetexte.bmp » sera codée dans le plan 2 (i.e. « vert »), le second pixel sera codé dans le plan 3 (i.e. « bleu »), le troisième dans le plan 1, le quatrième dans le plan 3, le cinquième dans le plan 2, le sixième dans le plan 1. Les pixels suivants seront codés en répétant la clef ; le septième sera codé dans le plan 2, le huitième dans le plan 3, etc.

On pourra créer deux fonctions crypt.m et decrypt.m pour crypter ou décrypter.

c. Etape 3 : Interface utilisateur

Au niveau de la forme, le programme devra incorporer une interface utilisateur transparente et se décomposer en plusieurs scripts et fonctions.

Un script principal devra demander à l'utilisateur les informations nécessaires au fonctionnement et lancer les différentes fonctions nécessaires :

- en premier lieu, il demandera si l'utilisateur veut crypter ou décrypter.
- en fonction de la réponse de l'utilisateur, il demandera les noms des fichiers images à crypter ou à décrypter.
- puis il demandera la clef de cryptage soit sous la forme d'une série (dans le cas du cryptage).
- Puis il lancera une fonction de cryptage (crypt.m) ou de décryptage (decrypt.m).
- Enfin, il en affichera le résultat (l'image résultante, ou de la clef codée) et le sauvegardera dans un fichier bmp.

IOVIS L3

Projet - Marche aléatoire

On s'intéresse à la marche aléatoire de quelques (1000) personnes ivres. Ces personnes sont bloquées sur un axe x et peuvent à chaque pas se déplacer vers la droite ou vers la gauche d'un pas de façon aléatoire (elles sont ivres). De plus les pas ne sont pas tous égaux et sont choisis de façon aléatoire entre 0 et ± 1 m.
On ajoute de plus des conditions aux limites en $-B$ et $+B$ que l'on précisera dans la suite. Toutes les personnes partent en $x = 0$.

Les barrières peuvent être de deux genres différents :

- Soit une barrière *absorbe* le marcheur, c'est à dire par exemple en $-B$ et $+B$ le marcheur tombe dans un trou et disparait de la simulation.
- Soit une barrière *refléchit* le marcheur, c'est à dire par exemple en $-B$ et $+B$ le marcheur arrive nez à nez avec un policier qui le force à faire demi tour.

De manière générale la position du marcheur ivre après le n^{ieme} pas est donnée par une suite de la forme :

$$x_{j+1} = x_j + s,$$

avec s le pas choisit de façon aléatoire entre -1 et $+1$ m.

1. (a) Creer une fonction Matlab 'marcheur.m' permettant de simuler la marche aléatoire de 1 marcheur alcoolique pendant P pas de taille s. La sortie de cette fonction sera une vecteur colonne qui représente la trajectoire d'un marcheur, c'est à dire sa position après le pas correspondant au numéro de la ligne. Dans cette premiere fonction, on ne prends pas de condition aux limites.

   (b) Modifier la fonction pour prendre en compte une barriere réflechissante en B et $-B$.

   (c) Modifier la fonction pour prendre en compte une barriere absorbante en B et $-B$. Une fois absorbé la valeur prise par la trajectoire du marcheur est NaN.

   (d) Créer une deuxieme fonction 'population.m' qui renvoie en sortie une matrice NxP où chaque colonne correspond à une trajectoire d'un marcheur différent.
   Les entrées de cette fonction sont :
   — N, la nombre de marcheurs
   — P le nombre de pas
   — s la taille des pas,
   — la variable *barriere* qui peut prendre 3 valeurs : *barriere=0 signfie pas de barierre, barriere=1 signifie barriere absorbante, barriere=-1 signifie barriere reflechissante*
   — B la position de la barriere.

   (e) Ajouter une sortie à cette fonction pour qu'elle puisse renvoyer un vecteur avec la position finale des N alcooliques.

2. Testez votre fonction 'population.m' en affichant la trajectoires de 20 marcheurs sans barriere, puis 5 marcheurs avec une barriere reflechissante, puis 50 marcheurs avec une barriere absorbante.
   Faites tous les autres tests que vous jugerez nécessaires pour verifier que votre fonction fonctionne correctement.

3. Statistiques :
   (a) Calculez la position moyenne des N marcheurs en fonction du temps (ou des pas)
   (b) Calculez l'ecart type de la position des N marcheurs en fonction du temps (ou des pas)
   (c) Est ce que la position moyenne et l'écart type sont modifiés en fonction du type de conditions aux limites ?
   (d) Affichez un histogramme de la position finale des marcheurs dans les 3 cas.
   (e) Calculez la durée de vie moyenne d'un marcheur pour des barrieres absorbantes. On pourra utiliser la fonction '$\sim$ isnan' pour compter le nombre de point non Nan dans une colonne.
   (f) Comment évolue cette durée de vie en fonction de la position de la barrière ? Pour cette question on pourra faire une boucle sur la question précédente
4. Question Bonus :
   (a) Ajouter la possibilité à la fonction de prendre en compte une barriere partiellement réflechissante en B et $-B$ c'est à dire que lorsque le marcheur arrive à la barrière il peut soit être absorbé, soit réfléchi de façon aléatoire.
   (b) Calculez la durée de vie.
   (c) Tracez le nombre d'alcooliques survivants en fonction du temps.
   (d) Quel est la densité de marcheurs en fonction du temps. Affichez cette donnée.

Projet Matlab : La planche de Galton.

Tom Darras - Quentin Glorieux

tom.darras@lkb.upmc.fr - quentin.glorieux@lkb.upmc.fr

La Planche de Galton est un dispositif permettant de réaliser des expériences de statistique. Elle est constituée de rangées de clous régulièrement espacés formant un triangle sur lesquel on fait tomber des billes. Lorsque qu'un bille touche un clou, elle a une chance sur deux d'aller à gauche, et une chance sur deux d'aller à droite. La bille finit sa trajectoire en tombant dans une des boîtes en bas de la planche. On s'intéresse à la répartition des billes dans les boîtes.

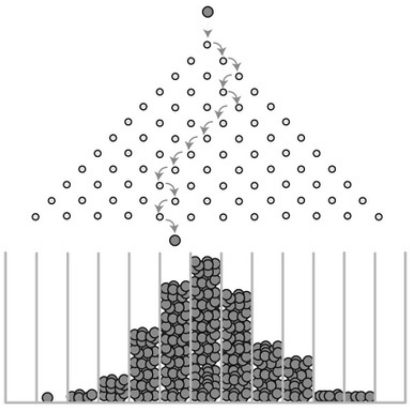


Figure 1: schéma d'une planche de Galton et de distribution des billes dans les boîtes.

1. Réaliser un script Matlab simulant la trajectoire d'une bille dans une planche de Galton constituée de N rangées de clous. On considérera pour l'instant que la bille à une probabilité $p = 1/2$ d'aller à gauche lorsque qu'elle tape un clou.

2. Répéter l'expérience pour un nombre *N_billes* de billes. Tracer l'histogramme des positions des billes.

3. Généraliser votre script en l'adaptant aux valeurs quelconques de p. Qu'observe-ton au niveau de l'histogramme lorsque l'on fait varier p ?

4. Par quelle loi statistique est régie la distribution des billes dans les boîtes ? Justifier que dans le cas p=1/2, cette distribution tend vers une loi normale lorsque le nombre de rangée est élevé et qu'il y a suffisamment de billes.

5. En utilisant votre simulation de la planche de Galton, illustrer le phénomène de régression vers la moyenne.

Projet Matlab : le jeu de la vie de John Conway.

Tangui Aladjidi - Quentin Glorieux

tangui.aladjidi@lkb.upmc.fr - quentin.glorieux@lkb.upmc.fr

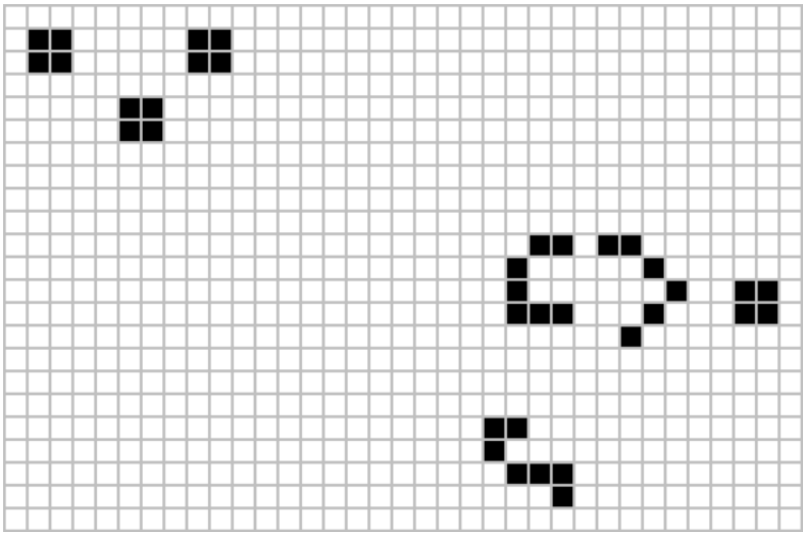


Figure 1: Exemple d'une partie du jeu de la vie. Les cases noires sont les cellules vivantes, les blanches sont mortes.

Le jeu de la vie est un "jeu à 0 joueur" selon son créateur John Conway. Il est un exemple très simple et fascinant de comportements émergeants. C'est ce qu'on appelle un automate cellulaire. Le jeu se joue classiquement sur une feuille de papier à carreaux, où l'on va noircir des cases. Voici les règles :

- Si une cellule morte a exactement 3 voisines, elle naît.
- Si une cellule vivante a 2 ou 3 voisines, elle le reste, sinon elle meurt.

Ces règles sont une version très simplifiée des règles qui régissent les vraies cellules. Si une cellule est totalement isolée, elle mourra. À l'inverse, si elle est entourée de trop de congénères, elle finira asphyxiée et mourra également. En revanche, s'il y a juste le bon nombre de voisins, les cellules se reproduisent.

À chaque tour, on applique les règles à chaque cellule pour obtenir la configuration suivante (la génération suivante).

I. INTRODUCTION

a. À partir de la configuration donnée en exemple, générer à la main la génération suivante.

b. Écrire un code qui initialise un tableau de N_y lignes et N_x colonnes, de type Booléen, dont tous les éléments valent "Faux". On va stocker l'état des cellules comme vivant = vrai, morte = faux. Vous pouvez commencer avec une petite grille (par exemple N_x=N_y=32). Choisissez ensuite au hasard une configuration de départ.

c. Générer une grille avec une proportion donnée de cellules vivantes et mortes (par exemple 20% de vivantes, 80% de mortes).

d. Vérifier en affichant le tableau dans une fenêtre graphique.

II. COMPTAGE DES VOISINS

a. Écrire une fonction qui prend en argument une position (i, j) d'une cellule et qui retourne le nombre de voisins vivants de cette cellule.

b. Que se passe-t-il lorsqu'on est près du bord aux positions i=0 ou j=0 par exemple ? Implémenter des conditions de bord périodiques : les indices des tableaux doivent être pris **modulo** N_x (pour les lignes) et N_y (pour les colonnes).

c. Vérifier le comportement de la fonction en partant d'une distribution aléatoire où l'on pourra facilement vérifier les résultats de la fonction en affichant le tableau.

III. MISE À JOUR DU TABLEAU

a. À partir de la fonction comptant le nombre de voisins, écrire une fonction qui prendra la grille à la génération N, et retournera la grille à la génération $N + 1$. *Indice : On pourra utiliser simplement une boucle sur les éléments de la grille et en fonction du nombre de voisins, décider si la cellule survit, meurt, ou naît.* On vérifiera sur le même petit tableau que précédemment.

b. Écrire un script qui prend un tableau initial, puis le met à jour N fois, en affichant le résultat dans une fenêtre graphique qui montrera l'évolution.

c. Sauvegarder le résultat sous la forme d'une vidéo à 5 images par seconde. *Indice : on pensera à regarder la documentation de la fonction* VideoWriter*, et on pourra utiliser* getframe(gcf) *afin de récupérer l'image du graphique.*

d. Que se passe-t-il quand on fait varier la proportion de cellules vivantes initiale ? Remarquez-vous des configurations particulières ? Décrivez les.

IV. CONFIGURATIONS PARTICULIÈRES

a. Créer des fonctions pour générer à une position arbitraire une configuration qui se répète toutes les n générations ($n \geq 2$).

b. Créer une fonction pour générer un "planeur" à une position arbitraire.

c. Créer une fonction pour générer un "canon à planeurs" à une position arbitraire.

V. BONUS

Partant d'une grille arbitraire, peut-on savoir si ou comment la partie se termine ? Pourquoi ?

IOVIS L3

Projet - Bataille

En utilisant des regles de la bataille que vous préciserez, simulez 1.000.000 de parties pour trouver le nombre de coups moyens d'une partie. Y'a t il des parties infinies ?