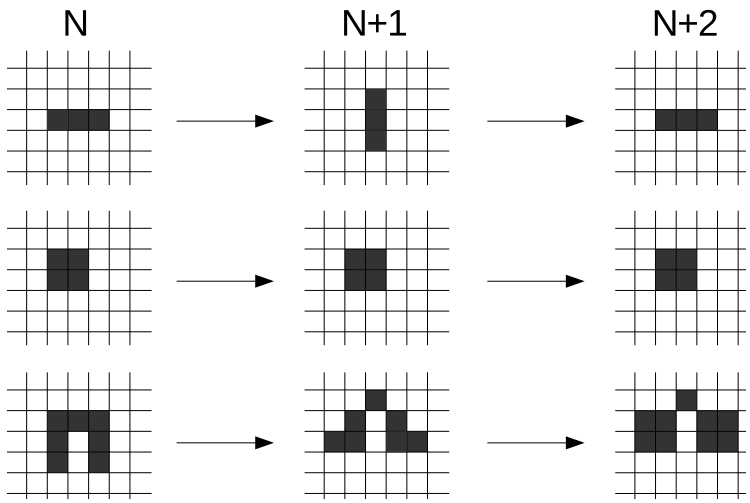
TP 5 - Corrigé partie 1
Jeu de la vie

Les solutions données dans ce corrigé ne sont bien sûr que des propositions, et sont sans nul doute perfectibles. Parfois on proposera même plusieurs solutions en discutant de leur pertinence.

1 Introduction

Q1 Après application des règles on obtient :



2 Implémentation

Q2 Dans le cas où N est pair, il faut faire un choix sur le centre de la grille : contentons-nous de prendre la partie entière de $\frac{N}{2}$, qui peut être obtenue en Python par `N//2`. Il peut être judicieux de commencer par l'affectation `c = N//2`, d'une part par soucis de lisibilité, d'autre part pour s'assurer que l'interpréteur Python ne calcule bien qu'une fois cette valeur.

Plutôt que d'écrire 7 instructions très proches, on pourrait être tenté de factoriser (par une boucle, ou en sélectionnant plusieurs cellules d'un coup), mais ici c'est sans doute se donner du mal pour rien. L'important est de procéder **méthodiquement**, pour n'oublier aucune cellule et éviter d'initialiser deux fois la même : par exemple en procédant colonne par colonne de gauche à droite et de haut en bas.

Listing 1 – init_jeu_u (1ere version)

```
1  def init_jeu_u(N) :
2      grille = np.zeros((N, N), dtype=float)
3      c = N//2
4
5      grille[c-1,c-1] = 1
6      grille[c-1,c] = 1
7      grille[c-1,c+1] = 1
8      grille[c,c+1] = 1
9      grille[c+1,c-1] = 1
10     grille[c+1,c] = 1
11     grille[c+1,c+1] = 1
12
13     return grille
```

Listing 2 – init_jeu_u (2e version)

```
1  def init_jeu_u(N) :
2      grille = np.zeros((N, N), dtype=float)
3      c = N//2
4
5      # On selectionne 3 cellules sur la meme colonne
6      grille[c-1,c-1:c+2] = 1
7      grille[c,c+1] = 1
8      grille[c+1,c-1:c+2] = 1
9
10     return grille
```

Listing 3 – init_jeu_u (3e version, avec une boucle)

```
1  def init_jeu_u(N) :
2      grille = np.zeros((N, N), dtype=float)
3      c = N//2
4
5      # On cree une liste d'abscisses et d'ordonnees pour
           parcourir les points
6      # (-1,-1), (-1,0), (-1,1), (0,1), (1,-1), (1,0), (1,1)
7      lX = [-1, -1, -1, 0, 1, 1, 1]
8      lY = [-1, 0, 1, 1, -1, 0, 1]
9
10     # zip permet de parcourir simultanement les listes lX et lY
11     for (x,y) in zip(lX, lY):
12         grille[c+x, c+y] = 1
13
14     return grille
```

Q3 Pas de difficulté : on parcourt la grille et on crée la liste des abscisses et des ordonnées des points à tracer. On pensera à effacer la figure courante avant d'afficher les points.

Listing 4 – afficher_jeu

```
def afficher_jeu(grille):
    X = []
    Y = []
    for i in range(1,grille.shape[0]-1):
        for j in range(1,grille.shape[1]-1):
            if (grille[i,j] != 0):
                X.append(i)
                Y.append(j)
    plt.clf()
    plt.axis([0, grille.shape[1], 0, grille.shape[0]])
    plt.plot(X, Y, 'ks')
```

Q4 Ici aussi on peut être tenté d'écrire plutôt une boucle mais au final il est sans doute plus sage et rapide d'écrire chaque voisine à la main. On proposera néanmoins les deux versions.

Il est important de comprendre pourquoi le fait de supposer que (i, j) ne désigne pas une cellule du bord simplifie grandement l'écriture de cette fonction. En effet si par exemple $i = N - 1$ il faudrait faire attention à ne pas tester les voisines de droite sous peine de « sortir »du tableau ! De même pour les bords gauche, haut et bas.

Listing 5 – compter_voisins (1ere version, en une ligne)

```
def compter_voisins(grille, i, j):
    return (grille[i-1,j-1] + grille[i-1, j] + grille[i-1,j+1] +
        grille[i,j-1] + grille[i,j+1] + grille[i+1,j-1] + grille
       [i+1, j] + grille[i+1,j+1])
```

Listing 6 – compter_voisins (2e version, avec une boucle)

```
def compter_voisins(grille, i, j):
    # on cree une liste d'abscisses et d'ordonnees pour
       parcourir les cellules voisines
    lX = [-1, -1, -1, 0, 0, 1, 1, 1]
    lY = [-1, 0, 1, -1, 1, -1, 0, 1]

    s = 0
    for (x,y) in zip(lX, lY):
        s += grille[i+x,j+y]
    return s
```

Q5 On demande que la fonction modifie la grille passée en argument et ne retourne rien. On voit qu'on a besoin d'effectuer une copie de la grille avant toute chose : l'état d'une cellule de génération $N+1$ dépend de l'état de ses voisines à la génération N, donc il faut garder une copie de la configuration N inchangée !

Faire attention à ne parcourir que l'intérieur de la grille, pour ne pas appeler la fonction `compter_voisins` sur les bords.

Listing 7 – iterer_jeu

```
1  def iterer_jeu(grille):
2      precG = grille.copy()
3
4      for i in range(1,grille.shape[0]-1):
5          for j in range(1,grille.shape[1]-1):
6              c = compter_voisins(precG, i, j)
7              if ((grille[i,j] == 1 and c == 2) or c == 3):
8                  grille[i,j] = 1
9              else:
10                 grille[i,j] = 0
```

Q6 Pas de difficulté ici.

Listing 8 – iterer_jeu modifiée

```
1  def iterer_jeu(grille, nb_iter, afficher = False):
2      for k in range(nb_iter):
3          precG = grille.copy()
4
5          for i in range(1,grille.shape[0]-1):
6              for j in range(1,grille.shape[1]-1):
7                  c = compter_voisins(precG, i, j)
8                  if ((grille[i,j] == 1 and c == 2) or c == 3):
9                      grille[i,j] = 1
10                 else:
11                     grille[i,j] = 0
12         if (afficher):
13             afficher_jeu(grille)
14             plt.title('${:3d}$'.format(k))
15             plt.pause(10**-15)
```