HEXAPAWN

1 Représentation Python

1. Rien de particulier ....

```
def init():
    return [[1,1,1],[0,0,0],[2,2,2]]
```

2. Il y a plusieurs façons de faire. Il est néanmoins préférable de faire une boucle décroissante pour que la présentation des parties soit la même que sur la feuille de papier.

```
def affiche(p):
    for i in range(len(p)):
        s=""
        for y in p[2-i]:
            s=s+str(y)+"␣"
        print(s)
```

2 Calcul des coups jouables et du gagnant

3. On parcourt tout simplement le plateau à la recherche des coordonnées des cases à j.

```
def pieces(j,p):
    L=[]
    for x in range(len(p)):
        for y in range(len(p[x])):
            if p[x][y]==j:
                L.append((x,y))
    return L
```

4. Il y a au plus trois déplacements possibles. Dans tous les cas, on vérifie que `pos[0]+d` est bien un indice de ligne correct. Pour les attaques, on prend grade à vérifie que `pos[y]` ± 1 est également un indice de colonne correct. Suivant la nature du coup (attaque/déplacement), on vérifie si la case est vide ou contient un pion adverse. On n'oublie pas d'effacer la position précédente du pion.

```
def coups_jouables_pion(j,pos,p):
    L=[]; d=3-2*j; adv=3-j
    if (pos[0]+d) in [0,1,2]:
        # avancement du pion
        if p[pos[0]+d][pos[1]]==0:
            p1=copier(p)
            p1[pos[0]+d][pos[1]]=j
            p1[pos[0]][pos[1]]=0
            L.append(p1)
        # prise en diagonale droite
        if pos[1] in [0,1] and p[pos[0]+d][pos[1]+1]==adv:
                p1=copier(p)
                p1[pos[0]+d][pos[1]+1]=j
                p1[pos[0]][pos[1]]=0
                L.append(p1)
        # prise en diagonale gauche
        if pos[1] in [1,2] and p[pos[0]+d][pos[1]-1]==adv:
                p1=copier(p)
                p1[pos[0]+d][pos[1]-1]=j
                p1[pos[0]][pos[1]]=0
                L.append(p1)
    return L
```

5. Il suffit de concaténer toutes les listes des coups possibles pour tous les pions du joueur actif.

```
def coups_jouables(j,p):
    pions=pieces(j,p)
    L=[]
    for pos in pions:
        L=L+coups_jouables_pion(j,pos,p)
    return L
```

6. On vérifie simplement les trois conditions de victoires. A noter que seule la condition de blocage nécessite de savoir quelle est le nom du joueur actif.

```
def gagnant(joueur,p):
    if 2 in p[0] or pieces(1,p)==[]:
        return 2
    if 1 in p[2] or pieces(2,p)==[]:
        return 1
    if coups_jouables(joueur,p)==[]:
        return (3-joueur)
    return 0
```

3 Parties au hasard

7. Voici un exemple d'implémentation avec une boucle `while` qui teste la présence d'un gagnant au début de chaque boucle. On n'oublie pas de changer de joueur à chaque tour de boucle.

```
def partie_hasard_affichee():
    p=init(); joueur=1
    affiche(p); print("␣")
    while(gagnant(joueur,p)==0):
        l=coups_jouables(joueur,p)
        p=random.choice(l)
        joueur=3-joueur
        affiche(p); print("␣")
    print("le␣gagnant␣est␣"+str(gagnant(joueur,p)))
```

8. On supprime les lignes 3 et 8 du code précédent et on remplace le `print` de la ligne 9 par un `return`.

```
def partie_hasard():
    p=init(); joueur=1
    while(gagnant(joueur,p)==0):
        l=coups_jouables(joueur,p)
        p=random.choice(l)
        joueur=3-joueur
    return gagnant(joueur,p)
```

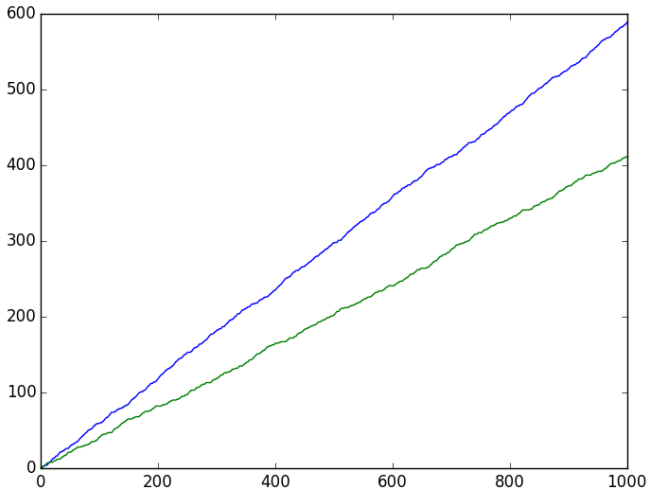
9. On lance 1000 parties avec un compteur de score pour chaque joueur qu'on met à jour après chaque partie. Les scores cumulés sont stockés au fur et à mesure.

```
c1=0; c2=0
L1=[0]
L2=[0]
for i in range(1000):
    g=partie_hasard()
    if g==1:
        c1+=1
    else:
        c2+=1
    L1.append(c1)
    L2.append(c2)

X=[i for i in range(len(L1))]
```

```
plt.clf()
plt.plot(X,L1)
plt.plot(X,L2)
```

La progression de chaque joueur est régulière, le joueur blanc (qui commence) gagnant environ 6/10 des parties.



4 Parties avec apprentissage

10. Voici une version naïve qui teste simplement si les éléments de `l` sont perdants avant de les stocker dans une nouvelle liste.

```
def elimine_coups_perdants(l):
    l2=[]
    for x in l:
        if not (x in perdants):
            l2.append(x)
    return l2
```

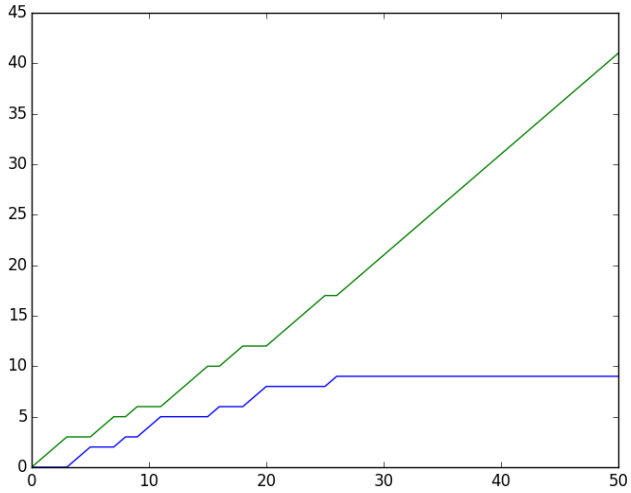
11. Par rapport à la version précédente :

- On ajoute une variable pour retenir l'avant-dernier coup joué.
- On ajoute un test pour savoir si la liste des coups non-perdants n'est pas vide. Dans le cas contraire, on utilise un `return None` pour interrompre la boucle (un peu sale, mais c'est ce que j'ai trouvé de plus simple à écrire). Enfin, on rajoute les ajouts dans `perdants` en cas de défaire du joueur 2.

```
def partie_hasard_affichee_apprenante():
    p=init(); pred=None; joueur=1
    affiche(p); print("␣")
    while(gagnant(joueur,p)==0):
        pred=copier(p)
        l=coups_jouables(joueur,p)
        if joueur==2:
            l=elimine_coups_perdants(l)
            if l==[]:
                print("position␣perdante␣pour␣joueur␣2")
                perdants.append(pred)
                return None
        p=random.choice(l)
        joueur=3-joueur
        affiche(p); print("␣")
    print("le␣gagnant␣est␣"+str(gagnant(joueur,p)))
```

```
17          if gagnant(joueur,p)==1:
18              perdants.append(pred)
```

12. Pour la modification, on supprime à nouveau les `print` et on ajoute un `return`. Les tests sur environ 50 parties donnent le graphe suivant :



Autrement dit, le joueur 2 gagne systématiquement une fois qu'il a perdu une petite dizaine de parties à peine ! En réalité, il dispose d'une stratégie gagnante, selon les principes expliqués dans le cours.