Ainpho

MPSI2 2016 2017
Lycée Charlemagne

Vendredi 6 janvier

Grand roman — Ainpho

Vous avez décidé d'écrire un roman épique/énorme. Vous tapez vingt pages par jours *(des pages du format de ce document)*. En combien de temps aurez vous rattrapé le grand Victor et ses Misérables *(ou Guerre et Paix de Tolstoï c'est la même taille)* ? En combien de temps aurez vous rattrapé le grand feuilleton des décimales de π ? En combien de temps aurez vous saturé une clef de 8Gigaoctets ? 5 pt.

Mystère — Ainpho

Voici deux procédures mystères. A vous de leur donner un nom en expliquant ce qu'elles font *(j'ai vu des sujets de concours proposer ça)*. 5 pt.

```
def mystere(n) :
....for k in range(2, n) : #et pas n+1
........if n%k == 0 :
............return(True)
....return(False)
```

```
def Scooby(N) :
....L = []
....n = 6
....while len(L) < N :
........if mystere(n) :
............L.append(n)
........else :
............L = []
........n = n+1
....return(L)
```

Que vont donner les exécutions suivantes :

`mystere(15)`	`mystere(2017)`	`Scooby(6)`	`Scooby(10)`	`Scooby(40)`

vous avez droit à un joker... 5 pt.

Halma/Dames chinoises — Ainpho

La variante du jeu de halma envisagée ici est un jeu de plateau qui peut se jouer à deux, trois, quatre ou même six *(et on peut aussi y jouer en solitaire)*. Les versions à deux, trois et six se pratiquent sur un plateau hexagonal dont les cases sont elles même des hexagones. On va envisager ici un jeu à deux joueurs.

Chaque joueur dispose de dix pions, disposés pour l'un sur les quatre premières lignes et pour l'autre sur les quatre dernières.

Les pions ont tous la même valeur *(comme en début de partie aux dames, mais ici, ils ne sont pas promus en quoi que ce soit d'autre)*. Les pions ne disparaissent pas au cours de la partie, même quand un pion saute au dessus d'eux *(contrairement au jeu de dames)*. Le but du jeu est d'amener ses pions le plus vite possible dans le coin opposé du jeu, là où étaient les pions de l'adversaire au départ. Le plus vite possible, ou pour le moins avant que l'adversaire n'ait fait la même chose en sens inverse.

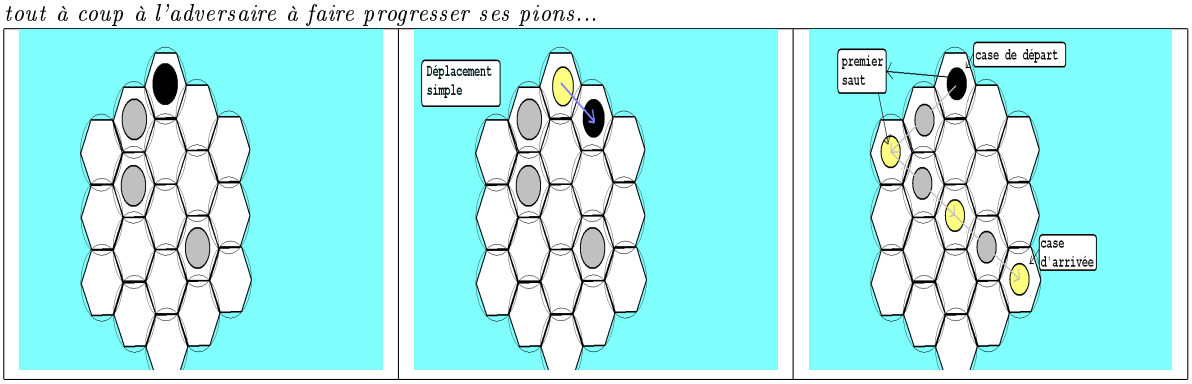
Les joueurs jouent à tour de rôle en déplaçant un pion à la fois.

Pour déplacer un pion, on peut

• l'amener sur une des six cases contigües *(il peut avancer, reculer, c'est sans importance)*

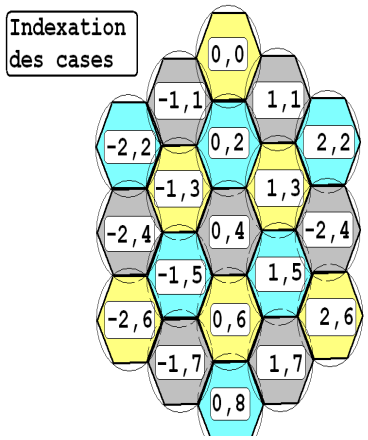
• le faire progresser en sautant : il doit sauter au dessus d'une case occupée et arriver sur une case vide, et il peut poursuivre sa route ainsi tant qu'il peut jouer à saute mouton *(ce qui là encore rappelle*

les dames).
Mais dans les sauts, ce qui fait l'originalité du jeu est que l'on n'élimine pas le pion par dessus lequel on a sauté, et surtout, la couleur du pion au dessus duquel on saute n'a aucune importance. C'est ce qui fait qu'une belle "échelle" construite pas vos soins pour amener rapidement des pions à l'autre bout du plateau peut servir tout à coup à l'adversaire à faire progresser ses pions...



Le but de notre exercice n'est évidemment pas de jouer au halma, ni même d'y joueur contre l'ordinateur. Il est de mettre en place quelques procédures pour concevoir un jeu de halma sur ordinateur.

Comment sont numérotées les cases ? Par deux indices : ligne et colonne. L'indice de ligne va de `0` à `16`, et l'indice de colonne peut aller de `-4` à `4` mais il y a des valeurs impossibles selon les lignes. Le schéma ci contre vous indique sur un plateau réduit comment les cases sont numérotées.



♯1 Votre première mission : CRÉER LA LISTE DE TOUTES LES CASES DU PLATEAU **[[0,0], [1, -1],...[16, 0]]**. 4 PT.

Question avant de programmer : COMBIEN DE CASES ? 1 PT.

Maintenant, on va faire mieux :

- CRÉEZ LA LISTE `L0` DES CASES OCCUPÉES PAR LES PIONS DU JOUEUR `0` 1 PT.
- CRÉEZ LA LISTE `L1` DES CASES OCCUPÉES PAR LES PIONS DU JOUEUR `1` 1 PT.
- CRÉEZ LA LISTE `Libres` DES CASES QUI RESTENT. 1 PT.

Comment va se matérialiser un coup joué ? On va modifier les listes. Si le pion du joueur `0` en `[2,2]` passe en `[1,3]` *(c'est légitime comme déplacement ?)* alors on `remove [2,2]` de la liste `L0`, on l'`append` à la liste `Libres`, on `remove [1,3]` de la liste `Libres` et on l'`append` à la liste `L0`. De sorte, `Libres` contient à tout instant les cases libres et `L0` et `L1` contiennent les positions des cases occupées par les joueurs.

♯1 ECRIVEZ UNE PROCÉDURE `possibles` QUI PREND EN ENTRÉE UNE POSITION `[i, j]` ET DONNE EN SORTIE [1] LA LISTE DES SIX CASES CONTIGÜES *(même avec des indices hors du plateau pour l'instant)*. 1 pt.

♯2 MODIFIEZ CETTE PROCÉDURE POUR QU'ELLE PRENNE EN ENTRÉE UNE POSITION `[i, j]` ET DONNE EN SORTIE LA LISTE DES CASES CONTIGÜES SUR LESQUELLES IL N'Y A PAS DE PION ET QUI SONT SUR LE PLATEAU. 3 pt.

♯3 MODIFIEZ CETTE PROCÉDURE POUR QU'ELLE PRENNE EN ENTRÉE UNE POSITION `[i, j]` ET DONNE EN SORTIE LA LISTE DES CASES CONTIGÜES SUR LESQUELLES IL N'Y A PAS DE PION, ET DES CASES SUR LESQUELLES ON PEUT ALLER EN UN SAUT *(à condition que la case du saut soit occupée par*

[1] `return` et pas `print`, on fait de la programmation, pas une conversation courtoise qui sera oubliée dans l'instant

un pion de `L0` ou `L1` et que la case d'arrivée soit libre). [4 pt.]

♯4 Programmeur expert : DONNEZ MÊME LA LISTE DES CASES AUXQUELLES IL PEUT ACCÉDER EN PLUSIEURS SAUTS. [8 pt.]

Remarque : il est inutile de vous préoccuper de choses comme "est ce que je risque d'arriver hors du plateau sur une case comme `[1, 2]` ou `[2, 1]`", puisque celle ci n'est pas dans `Libres` ... [2].

♯5 On veut savoir si un joueur a gagné. ECRIVEZ UNE PROCÉDURE `victoire` QUI PREND EN ENTRÉE LA LISTE `L0` DU JOUEUR `0` ET DONNE EN SORTIE `True` SI TOUS SES PIONS SONT ARRIVÉS DANS LA ZONE D'ARRIVÉE QUI LUI EST IMPOSÉE (LIGNES DU BAS). [2 pt.]

♯6 On veut une fonction d'évaluation qui indique si le coup que viendrait de joueur le joueur serait une bonne idée. ECRIVEZ UNE PROCÉDURE `evalue` QUI PREND EN ENTRÉE LA LISTE `L1` ET INDIQUE EN SORTIE LE NOMBRE DE DÉPLACEMENTS ÉLÉMENTAIRES (SANS SAUTS) QU'IL FAUDRAIT À TOUS SES PIONS POUR ARRIVER EN `[0, 0]`. [4 pt.]

On veut à présent s'occuper de la partie graphique. On ouvre une fenêtre avec `Tkinter`, on y crée un `Canvas` appelé `plateau` pour les dessins. On rappelle que le coin supérieur est numéroté `(0, 0)`, que l'unité est le pixel [3]. Il existe évidemment les méthodes `create_oval()` et `create_rectangle()`. Mais elles créent des ovales et des rectangles. Ici, il faut créer des hexagones. Il faudra donc utiliser `create_line()` : [4]. ECRIVEZ UNE PROCÉDURE `hexagone` QUI PREND EN ENTRÉE UNE ABSCISSE `x`, UNE ORDONNÉE `y` ET UN RAYON `R` ET TRACE L'HEXAGONE DE CENTRE `(x, y)` ET DE RAYON `R` *(`R` est le rayon du cercle inscrit contenant l'hexagone).* [3 pt.]

Conseil : si vous avez besoin plusieurs fois de la quantité $\sqrt{3}$, ne tapez pas des formules avec `sqrt(3)` partout. Validez d'abord `rac=sqrt(3)` et utilisez ensuite à chaque fois la constante `rac`. ; cela évitera à Python de faire, faire, re-faire et re-re-faire des calculs répétitifs inutiles.

Comme il faut bien faire du `SQL`, on imagine que des joueurs s'affrontent dans un tournoi de halma. On a alors une base de données faite de plusieurs tables :

table `Joueurs`							
champ	`Nom`	`Prenom`	`numero`	`sexe`	`date_naissance`	`adresse`	`telephone`
type	string	string	int	char	date	string	int
exemple 1			6				
exemple 2			7				

table `Parties`								
champ	`Joueur1`	`Joueur2`	`date`	`heure`	`vainqueur`	`Arbitre`	`plateau`	`NombreCoups`
exemple	6	7			7	12	5	124

Ecrivez une requête pour : [6 pt.]

compter le nombres de parties disputées dimanche 8 janvier
donner le pourcentage de filles inscrites
compter combien de parties a joué le joueur numéro 12
compter combien de parties a gagné le joueur nommé (utiliser une jointure)
donner la liste (nom, prénom) des dix participants les plus jeunes
calculer le nombre moyen de coups des parties jouées

[2] *sauf si bien sûr vous n'avez pas sû initialiser `Libres`*

[3] le `Canvas` sera donc "classiquement" de `width 800` et de `height 600` environ

[4] l'instruction `plateau.create_line(x, y, xx, yy)` trace une ligne du point de coordonnées `(x, y)` au point de coordonnées `(xx, yy)`

Rappel des mots clefs :
a in L : teste si l'élément a est dans la liste L
L.append(a) : ajoute l'élément a à la liste L
L.remove(a) : efface l'élément a de la liste L (erreur si absent de la liste)
L.sort() : trie la liste L
FROM table1 JOIN table 2 ON table1.indice=table2.index
SELECT ... ORDER BY champ LIMIT nombre
Les Misérables : 1800 pages de 3000 signes.
La recherche du temps perdu : 1million et demi de mots
Décimales de π connues : 13 000 milliards, commençant par 3,14159
1 octet = 8 bit, 1 caractère ASCII = 7 bit, 1 chiffre = 4 bit

MPSI2 2016 2017 Lycée Charlemagne

Correction

Ainpho

Un script en arithmétique. Ainpho

Que fait la procédure mystère ? Comme on a des modulo, on comprend que le `n` en entrée est un entier.
On passe en revue les `k` de `2` à `n-1`, et on regarde si ils divisent `n`.
Si l'un d'entre eux divise `n`, on sort et on retourne `True`.
Si on est allé au bout de la boucle, c'est qu'aucun n'a divisé `n`, on sort en retournant `False`.

	si `n` a un diviseur entre `2` et `n-1`	si `n` n'a aucun diviseur entre `2` et `n-1`
Bref,	`True`	`False`
	`n` est composé	`n` est premier

C'est tout le contraire d'un test *(bourrin mais simple)* de primalité.

En fait, c'est une fonction qui teste si un nombre est composé.

Si le `range` *de* `k` *était allé de* `0` *à* `n`*, on aurait eu une division par* `0`*.*
Si le `range` *de* `k` *était allé de* `1` *à* `n`*, on serait toujours sorti avec* `True` *dès* `k=1`
Si le `range` *de* `k` *était allé de* `1` *à* `n` *inclus, on serait sorti avec* `True` *au final avec* `k=n`

Enfin, les élèves qui n'ont rien compris et m'énervent proposent le script suivant :

```
def mystere(n) :
....for k in range(2, n) :
........if n%k == 0 :
............return("composé")
........else :
............return("pas composé")
```

En effet, avec ce script, on sort dès la première boucle.
Je pense aussi à l'erreur

```
def mystere(n) :
....for k in range(2, n) :
........if n%k == 0 :
............print("divisible")
........else :
............prit("pas divisible")
```

qui va afficher des tonnes de "*divisible*" et de "*pas divisible*" et l'utilisateur gentil et obéissant devra regarder si à un endroit dans la liste affichée il y a un "*divisible*" dans la liste des "*non divisible*".

Que fait la procédure suivante ? Elle prend en entrée un `N` qui doit être un entier, car à un moment, on le compare à la longueur d'une liste `L`.
On voit aussi un entier `n` qui commence à `6` et avance pas à pas *(*`n=n+1`*)*.
Si il est composé, on l'ajoute à la liste `L` *(qui va contenir des nombres composés)*.
Si il ne l'est pas, on remet la liste `L` à "zéro" *(elle ne contiendra donc que des nombres composés et jamais le moindre nombre premier)*.
Les entiers de la liste `L` sont donc des nombres premiers consécutifs *(quand un ne l'est pas, on efface tout et on recommence)*.
On complète donc des listes qu'on remet à zéro tant que la longueur de la liste n'a pas atteint `N`.

Bref, on crée une liste de N nombres composés consécutifs. (la première qui existe)

`mystere(15)`	`mystere(2017)`	`Scooby(6)`	`Scooby(10)`	`Scooby(40)`
True	False	[90, 91, 92, 93, 94, 95]	`[114, 115, 116, 117, 118, 119, 120, 121, 122, 123]`	joker

mais si vous y tenez : [15684, 15685, 15686, 15687, 15688, 15689, 15690, 15691, 15692, 15693, 15694, 15695, 15696, 15697, 15698, 15699, 15700, 15701, 15702, 15703, 15704, 15705, 15706, 15707, 15708, 15709, 15710, 15711, 15712, 15713, 15714, 15715, 15716, 15717, 15718, 15719, 15720, 15721, 15722, 15723]
dont voici même les factorisations :
[2^2.3.1307, 5.3137, 2.11.23.31, 3^3.7.83, 2^3.37.53, 29.541, 2.3.5.523, 13.17.71, 2^2.3923, 3.5231,
2.7.19.59, 5.43.73, 2^4.3^2.109, 11.1427, 2.47.167, 3.5233, 2^2.5^2.157, 7.2243, 2.3.2617,
41.383, 2^3.13.151, 3^2.5.349, 2.7853, 113.139, 2^2.3.7.11.17, 23.683, 2.5.1571, 3.5237,
2^5.491, 19.827, 2.3^4.97, 5.7.449, 2^2.3929, 3.13^2.31, 2.29.271, 11.1429, 2^3.3.5.131, 79.199, 2.7.1123, 3^2.1747]

Génération de la liste des cases. Ainpho

Il faut la liste des cases, comme sur l'exemple, mais avec plus de lignes et de colonnes. Et encore, peut on parler de lignes et colonnes avec nos hexagones.
On voit que ce n'est pas comme avec un tableau carré
`[[i,j] for j in range(N)] for i in range(N)]`
Pour un indice `j` donné, il n'y a que certaines valeurs de `i` possibles, négatives ou positives. Et en plus, il y a des considérations de parité.

L'élève peut abandonner estimant que c'est trop compliqué de voir la relation entre `i` et `j`. Ou alors il se demande si il doit prendre la solution "*bourrin*" qui consiste à taper in extenso la liste. Ce n'est pas une si mauvaise idée...

ligne 0					0,0					
ligne 1				-1,1		1,1				
ligne 2			-2,2		0,2		2,2			
ligne 3		-3,3		-1,3		1,3		3,3		
ligne 4	-4,4		-2,4		0,4		2,4		4,4	
ligne 5		-3,5		-1,5		1,5		3,5		
ligne 6	-4,6		-2,6		0,6		2,6		4,6	
ligne 7		-3,7		-1,7		1,7		3,7		
ligne 8	-4,8		-2,8		0,8		2,8		4,8	
ligne 9		-3,9		-1,9		1,9		3,9		
ligne 10	-4,10		-2,10		0,10		2,10		4,10	
ligne 11		-3,11		-1,11		1,11		3,11		
ligne 12	-4,12		-2,12		0,12		2,12		4,12	
ligne 13		-3,13		-1,13		1,13		3,13		
ligne 14			-2,14		0,14		2,14			
ligne 15				-1,15		1,15				
ligne 16					0,16					

Les cases en gras confirment l'exemple de l'énoncé sur un plateau plus petit.
On compte les cases : il y en a $2 \times (1+2+3+4+5) + (4+5+4+5+4+5+4)$ ce qui en fait $\boxed{61}$

Les valeurs possibles pour des damiers hexagonaux sont 1, 7, 19, 37, 61, 91, 127, 169 et ainsi de suite. On passe de 61 *à* 91 *en comptant le nombre de cases autour de notre modèle.*

Mais sinon, comment engendrer cette liste sans tout recopier ?
On engendre déjà un triangle du haut. Pour chaque valeur de `j` de `0` à `4` *(inclus)*, on crée une liste qui va de `-j` à `j` *(inclus)* avec un pas de `2` : `range(-j,j+1,2)`
Pour chaque `j` on va donc avoir `[[i,j] for i in range(-j, j+1,2)]`. On en est déjà à

```
L = []
for j in range(5) :
....L = L+[[i, j] for i in range(-j, j+1, 2)]
```

On va faire à peu près de même en fin de liste pour `j` de `12` à `16` *(inclus)* avec un i qui va de `j-16` à

16-j+1.
Ensuite, il y a les lignes du milieu qui alternent des `[[i,j] for i in range(-3, 3,2)]` et des `[[i, j] for i in range(-4, 4,2)]`
Même sans avoir étudié la théorie de l'information de Shannon, on se dit que le programme dans lequel on tape toutes les cases une par une n'est pas plus lourd que le programme avec des instructions en ligne...

Il y a quand même une solution "propre". On crée une liste qui pour chaque indice de ligne j donne l'indice i de première case :
`Lindex = [0, -1, -2, -3, -4, -3, -4, -3, -4, -3, -4, -3, -4, -3, -2, -1, 0]`
(par exemple : `Lindex[7]=-3` car la ligne 7 commence par `[-3, 7]` et finit par `[3, 7]`)
On connaît alors le `range` de la ligne `j` : `range(Lindex[j], 1-Lindex[j], 2)`
On tape alors le script suivant :

```
Lindex = [0, -1, -2, -3, -4, -3, -4, -3, -4, -3, -4, -3, -4, -3, -2, -1, 0]
L = [] #d'abord il n'y a rien
for j in range(17) : #il y a 17 lignes indexées de 0 à 16
....Lignej = [[i, j] for i in range(Lindex[j], 1-Lindex[j], 2))
....L = L+Lignej #on ajoute la nouvelle linge de bon format
print(L) #on vérifie en affichant
```

Et voilà le résultat :

```
[[0, 0], [-1, 1], [1, 1], [-2, 2], [0, 2], [2, 2], [-3, 3], [-1, 3], [1, 3], [3, 3], [-4, 4],
[-2, 4], [0, 4], [2, 4], [4, 4], [-3, 5], [-1, 5], [1, 5], [3, 5], [-4, 6], [-2, 6], [0, 6],
[2, 6], [4, 6], [-3, 7], [-1, 7], [1, 7], [3, 7], [-4, 8], [-2, 8], [0, 8], [2, 8], [4, 8],
[-3, 9], [-1, 9], [1, 9], [3, 9], [-4, 10], [-2, 10], [0, 10], [2, 10], [4, 10], [-3, 11],
[-1, 11], [1, 11], [3, 11], [-4, 12], [-2, 12], [0, 12], [2, 12], [4, 12], [-3, 13],
[-1, 13], [1, 13], [3, 13], [-2, 14], [0, 14], [2, 14], [-1, 15], [1, 15], [0, 16]]
```

61 éléments, c'est bon.

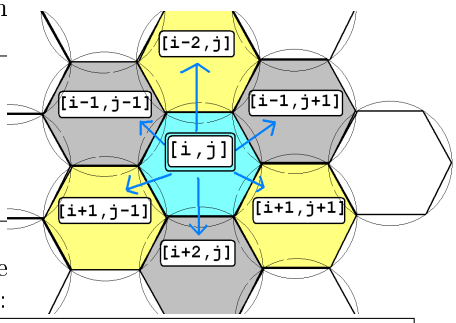
On a crée une liste de toutes les cases. On va la ventiler en trois listes :

```
Lindex = [0, -1, -2, -3, -4, -3, -4, -3, -4, -3, -4, -3, -4, -3, -2, -1, 0]
L0 = [] #on va créer la liste du joueur 0
for j in range(4) : #il a les quatre premières lignes
....Lignej = [[i, j] for i in range(Lindex[j], 1-Lindex[j], 2))
....L0 = L0+Lignej #des lignes de taille croissante
Libres = [] #on va créer la liste des cases libres
for j in range(4, 13) :
....Lignej = [[i, j] for i in range(Lindex[j], 1-Lindex[j], 2))
....L0 = L0+Lignej
L1 = [] #on va créer la liste du joueur 1
for j in range(13, 17) :
....Lignej = [[i, j] for i in range(Lindex[j], 1-Lindex[j], 2))
....L1 = L1+Lignej
print(L0, L1, Libres)
```

ligne 0					*0,0*					
ligne 1				*-1,1*		*1,1*				
ligne 2			*-2,2*		*0,2*		*2,2*			
ligne 3		*-3,3*		*-3,-1*		*-3,1*		*3,3*		
ligne 4	-4,4		-2,4		0,4		2,4		4,4	
ligne 5		-3,5		-1,5		1,5		3,5		
ligne 6	-4,6		-2,6		0,6		2,6		4,6	
ligne 7		-3,7		-1,7		1,7		3,7		
ligne 8	-4,8		-2,8		0,8		2,8		4,8	
ligne 9		-3,9		-1,9		1,9		3,9		
ligne 10	-4,10		-2,10		0,10		2,10		4,10	
ligne 11		-3,11		-1,11		1,11		3,11		
ligne 12	-4,12		-2,12		0,12		2,12		4,12	
ligne 13		-3,13		-1,13		1,13		3,13		
ligne 14			-2,14		0,14		2,14			
ligne 15				-1,15		1,15				
ligne 16					0,16					

Gestion du jeu. Ainpho

On part d'une case `[i, j]`. On regarde les cases voisines sur un plateau infini. On a donc sans effort :




```
def voisins(case) :
....i, j = case[0], case[1]
....V = [[i-2,j], [i-1,j-1], [i-1,j+1], [i+1,j-1],
........[i+1,j+1], [i+2,j]]
....return(V)
```

Là, on retourne une liste de cases dont certaines sont invalides. Pour ne garder que les cases qui sont dans `Libres`, on peut faire de l'intersection d'ensembles, Python sait faire. Mais on propose :

```
def voisins(case) :
....i, j = case[0], case[1]
....Tous = [[i-2,j], [i-1,j-1], [i-1,j+1], [i+1,j-1], [i+1,j+1], [i+2,j]]
....V = []
....for position in Tous :
........if position in Libres :
............V.append(position)
....return(V)
```

dans la nouvelle liste `V`, on n'accepte que les cases libres, donc à la fois sur la plateau, et sans pion. Si ensuite on veut les cases accessibles par saut :

```
def voisins(case) :
....i, j = case[0], case[1]
....Tous = [[i-2,j], [i-1,j-1], [i-1,j+1], [i+1,j-1], [i+1,j+1], [i+2,j]]
....V = []
....for position in Tous :
........if position in Libres :
............V.append(position)
........if position in L0+L1 :
............apressaut = [2*position[0]-i, 2*position[1]-j]
............if apressaut in Libres :
................V.append(apressaut)
....return(V)
```

Cette fois, si la case est libre, on la prend. Si la case est occupée par un pion *(fusion des deux listes `L0` et `L1` puisque la couleur des pions n'importe pas pour sauter)*, on regarde la case d'arrivée du saut et on s'interroge : est elle libre ?
Comment on a calculé la case d'arrivée du saut ? On partait de `[i, j]`, on sautait par-dessus `[i', j']`, on arrivait donc en `[i'+(i'-i), j'+(j'-j)]` *(relation de Chasles, symétrie centrale, géométrie de quatrième, voir l'exercice sur les TéléTubies sauteurs dans un devoir précédent).*

Pour itérer les sauts, c'est bien plus compliqué, d'autant qu'il faut éviter le schéma où le pions pourrait passer son temps à sauter *(il lui suffit de faire des allers-retours de mouton au dessus d'un pion entouré de deux cases libres)*. Il faudrait donc tenir une liste *(ou pile)* des cases déjà visitées et mettre en boucle une procédure inspirée de

```
if position in L0+L1 :
....apressaut = [2*position[0]-i, 2*position[1]-j]
....if apressaut in Libres :
........V.append(apressaut)
```

mais en indexant les positions occupées au fil des sauts.

Evaluation de victoire/gain. Ainpho

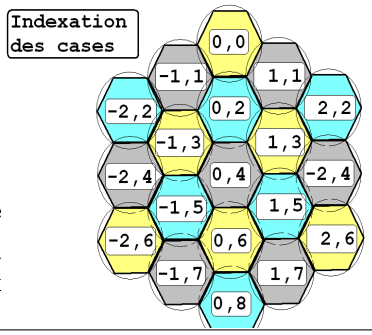
Pour savoir si le joueur `0` a gagné, il suffit de regarder si tous ses pions sont au delà de la ligne `13` *(incluse)*. On prend donc un par un les pions de la liste. Si l'un d'entre eux a un numéro de ligne pas assez grand, on sort tout de suite, sans même regarder les autres pions et on répond `False`. Si on a passé avec succès tous les tests, alors c'est que tous les pions sont bien placés, et on répond `True`.

```
def test_victoire(L) :
....for pion in L :
........if pion[1]<13 :
............return(False)
....return(True)
```

C'est simple et propre, sans aucun `else`. Vous savez, j'adore *(avec une petite tendresse ironique)* vos programmes en

```
if test  :
....n = n+1
else  :
....n = n
```

Pour l'évaluation de la somme des distances au point `[0, 0]`, il faut déjà regarder pour un pion en `[i, j]` combien il lui faut de déplacements élémentaires.
On observe que la somme $|i| + |j|$ est toujours paire. Tiens tiens. Et la quantité $\frac{|i|+|j|}{2}$ donne exactement le nombre de déplacements. On note qu'à chaque déplacement, i ou j changent. Soit un seul change de 2 unités, soit les deux changent d'une unité.




```
def Somme_distances(L) :
....S = 0
....for pion in L :
........S = S+abs(pion[0])+abs(pion[1])
....return(S/2)
```

Oui, autant tout sommer et ne diviser par 2 qu'à la fin. On s'épargne ainsi des divisions successives. Encore une réflexe de programmeur...

Hexagones graphiques. Ainpho

C'est en fait une question de géométrie. On a un centre en `[x, y]`, et un rayon `R`. On doit placer les six sommets de l'hexagone :

	$\left(x-\frac{R}{2}, y-\frac{R.\sqrt{3}}{2}\right)$		$\left(x+\frac{R}{2}, y-\frac{R.\sqrt{3}}{2}\right)$	
$\left(x-R,\ y\right)$		$\left(x,\ y\right)$		$\left(x+R,\ y\right)$
	$\left(x-\frac{R}{2}, y+\frac{R.\sqrt{3}}{2}\right)$		$\left(x+\frac{R}{2}, y+\frac{R.\sqrt{3}}{2}\right)$	

Si si, les signes plus et moins devant l'ordonnée sont corrects, puisque la ligne d'indice 0 est en haut.
On commencera par définir les constantes $R.\sqrt{3}/2$ et $R/2$ *(importer* ***math*** *)*
On définira ensuite les six sommets ***(sous forme de liste)***.
On créera une procédure pour tracer une ligne entre deux points pour ne pas répéter trop de fois `plateau.create_line(...)`.
Il ne restera qu'à tracer les six segments :

```
def trait(P, Q) :
....plateau.create_line(P[0], P[1], Q[0], Q[1])
R2, R3 = R/2, R*sqrt(3)/2
S = [[x+R, y], [x+R2, Y-R3], [x-R2, y-R3], [x-R, y], [x-R2, y+R3], [x+R2, y+R3]]
for k in range(6) :
....trait(S[k-1], S[k])
```

et voilà, tout est fait en une fois, et si on veut modifier des paramètres, c'est facile.
Pourquoi avoir défini une sous-procédure trait ? parce que comme ça, si on veut modifier la largeur ou la couleur des traits, il y a une seule ligne où il faut le faire.
Et aussi en vertu du pricipe de base en programmation :
si une même instruction revient plus de trois fois, fais en une sous-procédure
Pourquoi avoir crée `R2` et `R3` : pour le lisibilité et pour qu'il ne pose pas six fois sa division par 2 *(un ordinateur plus bête que vous : si on lui redemande un calcul déjà fait, il ne regarde pas dans ses brouillons, il le refait).*

Bases de données. Ainpho

compter le nombres de parties disputées dimanche 8 janvier

```
SELECT COUNT(*) FROM Parties WHERE DATE = "8/01/17"
```
(voir le format de dates pour être sûr)

donner le pourcentage de filles inscrites

```
SELECT COUNT(*) FROM Joueurs WHERE sexe = 'F' OR sexe = 'F'
/ SELECT COUNT(*) FROM Joueurs
```

compter combien de parties a joué le joueur numéro 12

```
SELECT COUNT(*) FROM Parties WHERE Joueur1 = 12 OR Joueur2 = 12
```

compter combien de parties a gagné le joueur nommé *(utiliser une jointure)*

```
SELECT COUNT(*) FROM Parties JOIN Joueurs
........ON Parties.vainqueur = Joueurs.numero
............WHERE Joueurs.Nom='Nguyen'
```

donner la liste *(nom, prénom)* des dix participants les plus jeunes

```
SELECT Nom, Prenom FROM Joueurs ORDER BY date_naissance DESC LIMIT 10
```
le tri des dates se fait a priori est ascendant

calculer le nombre moyen de coups des parties jouées

```
SELECT AVG(NombreCoups) FROM Parties
....WHERE not(vainqueur='')
```

Un grand roman. Ainpho

Une page de ce texte, c'est environ 90 signes par ligne et 45 lignes environ.

On effectue : 4 000 signes par page.
Une page de livre "*normal*" c'est trois mille signes *(sauf si l'éditeur est un escroc)* ; une page dans l'édition payée à un auteur, c'est 1 500 signes.
Vous tapez donc 80 000 signes par jour *(ça va les petits doigts, il n'en a pas marre le clavier ?)*
Les Misérables, c'est 5 400 000 signes. En soixante dix jous, c'est plié.
(et moi qui me lis/relis les Misérables depuis plus d'un an, je devrais avoir fini... Sinon, "Marienbad my love" de Mark Leach c'est 17 000 000 *de mots... il faut combien de temps pour écrire ça !)*
Et les décimales de π ? Là, il y en a bien plus.
Le quotient donne 162 500 000 jours... Un peu plus de quatre cent mille ans...
Là, je crois que je ne me lancerai jamais dans la lecture.

Chaque jour, vous noircissez | 20 pages | 80 000 caractères | 560 000 bits | 70 000 octets |
Une clef USB c'est 8.2^{30} octets *(en arrondi décimal* 8.10^9*).*
On effectue le quotient, vous avez 15 000 jours devant vous. 420 ans pour la remplir bon courage.
Une clef 8Giga, c'est 2000 *titre en MP3,* 15 *films, combien de livres ?*