TD n°2 d'algorithmique : Programmer un jeu sur la calculatrice (suite)

La programmation des jeux est une activité stimulante, dont la nature est très différente du jeu qu'elle met en scène. Les mathématiques ne sont pas toujours présentes et quand elles le sont, elles ne sont souvent pas bien compliquées, néanmoins les programmes officiels mettent de plus en plus les algorithmes au centre de l'enseignement des mathématiques et c'est tant mieux.

1. Le petit bac

Le rôle de la calculatrice va peut-être se limiter au choix de la lettre. On ne va pas mettre un dictionnaire dans la toute petite mémoire de notre Numworks, et donc on ne va pas lui demander de vérifier nos mots. Par contre, on peut saisir les catégories en début de jeu et, pour chaque lettre, saisir nos propositions. Cela remplacera le sempiternel tableau griffonné au dos du contrôle de maths... Une autre fonction que l'on peut confier à la calculatrice est le compte à rebours de la durée impartie pour la recherche, mais cela paraît difficilement conciliable avec la fonction de saisie.

🗝 Pour programmer ce jeu, il faut définir la fonction ou les fonctions que l'on souhaite confier à la calculatrice : option n°1 : choix de la lettre+saisies (catégories et propositions) ; option n°2 : choix de la lettre+compte à rebours. Dans tous les cas, je suppose que l'on voudra se passer de l'environnement graphique, la console Python suffisant.

J'ai choisi, dans cette première version, un programme qui donne la lettre (en vérifiant qu'elle n'a pas été donné déjà puisqu'il semble que cela soit une règle implicite de ce jeu), qui propose 10 catégories par défaut en offrant la possibilité de les changer au départ, et enfin qui enregistre les 10 propositions. Les propositions sont modifiables autant qu'on le souhaite, car à la fin de la saisie des 10 propositions on peut revoir notre liste de propositions, en changeant les noms qui ne nous conviennent pas ou en passant (la touche « esc », à côté de la touche « ok ») se révèle utile ici, de façon imprévue).

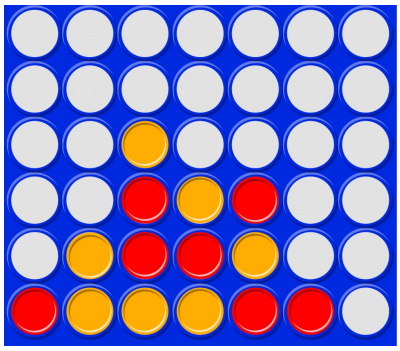
```
from random import *
def pb(s):
  categories=["animal", "prenom", "pays/ville",
"metier", "fleur/plante", "fruit/legume",
"produit/marque", "ecrivain/cineaste",
"sport", "vehicule"]
  lettres_deja=[]
  seed(s)#pour modifier la suite aleatoire
  for i in range(len(categories)):
    c=input("categorie {} : {} (esc:passe)".format(i+1,categories[i]))
    if c!="":categories[i]=c
  c=input("pret?")
  while c!="0":
    proposition=len(categories)*[""]
    l=randint(1,26)
    while l in lettres_deja:
      l=randint(1,26)
    print("la lettre est {}".format(chr(l+64)))
    lettres_deja.append(l)
    for i in range(len(categories)):
      proposition[i]=input("{} : ".format(categories[i]))
    c=input("esc:revoir,1:lettre,0:fin?")
    while c=="":
      for i in range(len(categories)):
        c=input("{} : {} ".format(categories[i],proposition[i]))
        if c!="":proposition[i]=c
      c=input("esc:revoir,1:lettre,0:fin?")
```

```
deg                PYTHON
>>> from petit_bac import *
>>> pb(12)
categorie 1 : animal (esc:pass
categorie 2 : prenom (esc:pass
categorie 3 : pays/ville (esc:
categorie 4 : metier (esc:pass
categorie 5 : fleur/plante (es
categorie 6 : fruit/legume (es
categorie 7 : produit/marque (
categorie 8 : ecrivain/cineast
categorie 9 : sport (esc:passe
categorie 10 : vehicule (esc:p
pret?
la lettre est M
animal : mante
prenom : marcel
pays/ville : marseille
metier : menuisier
fleur/plante : manguier
fruit/legume : melon
produit/marque : mnms
ecrivain/cineaste : musset
sport : musculation
vehicule : moto
esc:revoir,1:lettre,0:fin?
animal : mante
prenom : marcel
pays/ville : marseille
metier : menuisier
fleur/plante : manguier
fruit/legume : melon
produit/marque : mnms mercedes
```

Le fonctionnement est correct, on peut changer les catégories, on peut entrer les propositions et les changer ultérieurement. En entrant 0 comme réponse à la question « esc:revoir,1:lettre,0:fin? », le programme s'arrête ; en entrant 1, une nouvelle lettre est proposée et en utilisant la touche « esc » on peut revoir notre copie. Je ne comprend pas bien pourquoi, lorsqu'on n'entre rien dans un input, on ne peut pas le faire avec la touche « EXE », par contre, en utilisant la touche « esc », cela revient à entrer rien... Mystère, mais c'est finalement assez pratique. La programmation est souvent une science opportuniste : on bricole quelque chose qui marche avec les moyens du bord (du moment que ça marche...), quitte à améliorer les choses par la suite.

Des propositions pour l'amélioration : déjà, on pourrait vérifier que les mots entrés commencent bien par la bonne lettre. Éventuellement, on pourrait commencer le mot en entrant cette première lettre. Ensuite, on peut souhaiter une comptabilité des points gagnés : bien sûr, comme les mots sont vérifiés par l'utilisateur à la fin, on peut lui laisser le soin d'entrer le score, par exemple en remplaçant le 1 attendu pour une nouvelle lettre par le score qui serait alors ajouté au score précédent. Dans ce cas il faut modifier le code de sortie (mettre « . » par exemple pour sortir).

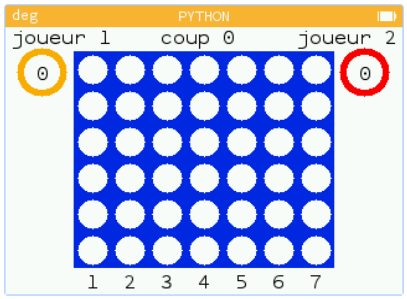
2. Puissance 4



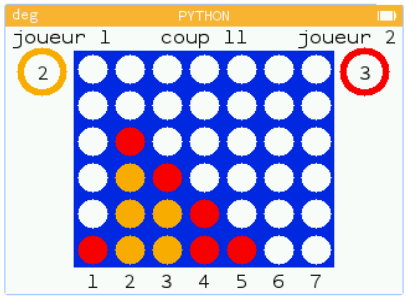
C'est une demande répétée, comme le jeu *snake* qui paraît difficile à réaliser à cause du mouvement qui s'ajoute aux nécessaires *input*. Contrairement à *snake*, puissance 4 est simple à programmer : il y a une interaction qui se limite à entrer la colonne, à tour de rôle, des pions jaunes et des rouges. Car puissance 4 se joue à 2. Il faut sans doute imaginer que le jeu se joue en plusieurs manches, ne serait-ce que deux manches gagnantes (les jaunes commencent la 1ère, les rouges la 2de et, si il y a une belle, on tire au sort celui qui commence). Les apprentis experts en intelligence artificielle pourront s'exercer à programmer un jeu contre la calculatrice.

🗝Pour programmer ce jeu, il faut dessiner le plateau 7×6. Le jeu, ensuite, doit enregistrer les positions des pions dans chaque case et leur couleur, et doit vérifier la présence d'un ou plusieurs alignements de 4 pions d'une couleur. On peut ajouter un affichage du score de chaque joueur si on prévoit plusieurs manches.

```
from kandinsky import *
from math import *

def rectangle(x,y,c,lon,lar):
  for i in range(lon):
    for j in range(lar):
      set_pixel(x+i,y+j,c)

def cercle(x0,y0,r,c,e):
  for i in range(2*e):
    xd=x0-int((r-i*0.5)/sqrt(2))
    xf=x0+int((r-i*0.5)/sqrt(2))
    for x in range(xd,xf+1):
      x1=x
      y1=y0+int(sqrt((r-i*0.5)**2-(x-x0)**2))
      set_pixel(x,y1,c)
      for j in range(3):
        x2=x0+y1-y0
        y2=y0+x0-x1
        set_pixel(x2,y2,c)
        x1,y1=x2,y2

  def plateau():
    rectangle(55,20,color(0,42,224),210,180)
    for i in range(42):
      cercle(70+30*(i//6),35+30*(i%6),12,color(255,255,255),12)
    for i in range(7):
      draw_string(str(i+1),65+30*i,203)

  def placer(col,lig,j):
    coul=[color(255,174,0),color(255,0,0)]
    cercle(70+30*(col),35+30*(lig),12,coul[j-1],12)

  def afficher(c):
    draw_string("coup {}".format(c),125,0)
    return c

  def modifier(s):
    draw_string(str(s[0]),25,30)
    draw_string(str(s[1]),285,30)
    return s

def controler(l):
  for lig in range(6):
    H=[]#control horizontal
    for col in range(7):
      H.append(l[col][lig])
    for pos in range(4):
      if H[pos]!=0 and H[pos]==H[pos+1] and H[pos]==H[pos+2] and H[pos]==H[pos+3]:
        return H[pos]
  for col in range(7):#control vertical
    for lig in range(3):
      if l[col][lig]!=0 and l[col][lig]==l[col][lig+1] and l[col][lig]==l[col][lig+2] and l[col][lig]==l[col][lig+3]:
        return l[col][lig]
  for col in range(4):#control oblique1
    for lig in range(3):
      if l[col][lig]!=0 and l[col][lig]==l[col+1][lig+1] and l[col][lig]==l[col+2][lig+2] and l[col][lig]==l[col+3][lig+3]:
        return l[col][lig]
  for col in range(4):#control oblique2
    for lig in range(3):
      if l[6-col][lig]!=0 and l[6-col][lig]==l[5-col][lig+1] and l[6-col][lig]==l[4-col][lig+2] and l[6-col][lig]==l[3-col][lig+3]:
        return l[6-col][lig]
  return 0
```




```
def p():
  plateau()
  draw_string("joueur 1",5,0)
  draw_string("joueur 2",235,0)
  cercle(29,37,20,color(255,174,0),5)
  cercle(289,37,20,color(255,0,0),5)
  score=[0,0]
  modifier(score)
  L=[6*[0],6*[0],6*[0],6*[0],6*[0],6*[0],6*[0]]
  ligne=False
  joueur=1
  cp=0
  afficher(cp)
  while ligne==False:
    place,n=0,0
    print(L)
    while n==0 or n>7 or place>5:
      n=int(input())
      while L[n-1][place]!=0:
        place+=1
    L[n-1][place]=joueur
    placer(n-1,5-place,joueur)
    cp+=1
    afficher(cp)
    fin=controler(L)
    if fin==0:
      joueur=(joueur)%2+1
    else:
      score[fin-1]+=1
      modifier(score)
      ligne=True
      e=int(input("nouveau? (1:oui,0:non)"))
      if e==1:
        cp=0
        joueur=joueur%2+1
        plateau()
        L=[6*[0],6*[0],6*[0],6*[0],6*[0],6*[0],6*[0]]
        ligne=False
```



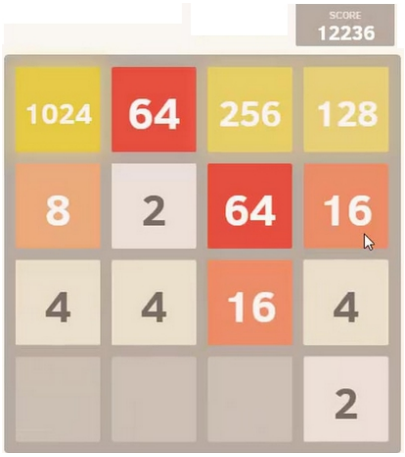
J'ai dit que c'était simple à programmer, mais tout est relatif... Disons que les entrées sont simples : alternativement on entre un chiffre pour la colonne. On peut comptabiliser les pions dans une colonne pour

savoir où positionner le nouveau pion, mais il faut aussi garder la mémoire de la couleur des pions... J'utilise donc une liste de 7 listes pour les 7 colonnes. Dans chacune de ces listes j'écris 1 ou 2 selon la couleur du jeton entré. Au début, si le joueur 1 entre son jeton dans la colonne centrale, la liste sera [[],[],[],[1],[],[],[]]. Si le joueur 2 met son jeton dans la colonne 5, la liste devient [[],[],[],[1],[2],[],[]], si le joueur 1 riposte en mettant son jeton dans la colonne 5, la liste devient [[],[],[],[1],[2,1],[],[]], etc. L'état de la liste après le 14ème coup (voir l'illustration) serait enregistré sous la forme [[2],[1,1],[1,2,2,1],[1,2,1],[2,1,2],[2],[]].
Jusque là c'est simple. Il faut ensuite prévoir le module qui teste toutes les directions d'alignement : les 6 lignes, les 7 colonnes et les 12 diagonales (6 dans chaque direction). Pour cela, il est préférable d'avoir des 0 dans les cases vides plutôt que des listes à géométrie variable. Car si l'on veut tester une ligne, on va lire le contenu de la ligne lig de la colonne col, en utilisant la syntaxe L[col][lig] qui provoque une erreur si il n'y a rien dans cet emplacement. C'est donc une liste L comme ceci qu'on aura après le 14ème coup :

[[2,0,0,0,0,0],[1,1,0,0,0,0],[1,2,2,1,0,0],[1,2,1,0,0,0],[2,1,2,0,0,0],[2,0,0,0,0,0],[0,0,0,0,0,0]]

On doit se pencher sur le nombre de vérifications à effectuer pour n'en oublier aucune : il y a 6 lignes et 4 possibilités d'alignement par ligne (les colonnes 1-2-3-4, 2-3-4-5, 3-4-5-6 ou 4-5-6-7), 7 colonnes et 3 possibilités d'alignement par colonne (les lignes 1-2-3-4, 2-3-4-5 ou 3-4-5-6). Pour les diagonales, il y a 6 droites dans 2 directions, mais cela fait 12 possibilités d'alignement, les points de départ étant situés dans un rectangle couvrant 3 lignes et 4 colonnes.

3. 2048



Ce jeu récent (créé par Gabriele Cirulli en mars 2014) a un fondement mathématique assez intéressant : il est bâti sur les puissances de 2, les nombres écrits sur deux plaques identiques peuvent fusionner et devenir une puissance de 2 supérieure, par exemple 2+2=4, 4+4=8, 16+16=32, …, 1024+1024=2048. Le but du jeu est d'atteindre la plaque 2048 sachant que souvent on se trouve bloqué avant, mais on peut continuer à jouer ensuite pour atteindre le meilleur score possible. Le plateau reçoit une nouvelle plaque 2 ou 4 sur une case libre, à chaque tour, selon une loi de probabilité immuable (je crois qu'il s'agit d'une probabilité de 1 sur 7 pour l'apparition des 4). Les déplacements se font dans une des 4 directions, à condition que des plaques peuvent se déplacer, pas forcément en fusionnant. Le jeu de l'illustration peut être déplacé vers le bas (les 3 colonnes de gauche descendront), vers la droite (les deux 4 fusionneront), vers la gauche (en plus de la fusion des 4, il y aura déplacement du 2sur la ligne du bas), mais pas vers le haut.

🗝Pour programmer ce jeu, il faut dessiner le plateau. Au départ, il y a deux plaques tirées selon les règles immuables (on prendra 1 chance sur 7 d'avoir un 4 ou bien on effectuera une statistique sur le jeu officiel à l'adresse https://gabrielecirulli.github.io/2048/). Le dynamisme est assuré par l'entrée d'un chiffre, selon la disposition du clavier (vers la gauche:4, la droite:6, le haut:8 et le bas:2). Lorsqu'il y a trois plaques identiques sur une ligne, la fusion se fait entre les plaques les plus proches du point d'arrivée. Le score cumule les plaques qui fusionnent.

Ici, le plateau est facile à dessiner, un quadrillage 4×4 ferait l'affaire. J'ai préféré essayer de coller au plus près au design du jeu officiel en dessinant des carrés gris clair dans un grand carré gris. Dessiner les plaques avec les nombres n'est pas très difficile non plus, si on sait où les dessiner. Le problème vient ici du mouvement qui est assez complexe à modéliser.
J'ai simplifié le problème en ramenant les 4 sens différents à un seul traitement et, comme ce traitement ne concerne qu'une seule ligne, je n'ai qu'une seule ligne à traiter. Ce traitement est effectué par la fonction move() que j'ai écrite dans un fichier « *utilitaires.py* » qu'il faut importer dans le programme principal. La dynamique d'une ligne n'est pas évidente à traiter : je n'ai rien trouvé de mieux que de distingue tous les différents cas, et il y en a une vingtaine, rien que pour une ligne. C'est un peu fastidieux et il y a sans doute une façon de faire qui simplifierai le travail, mais cela fonctionne. Au passage, il faut que cette fonction move() nous apporte le nombre a ajouter au score (c'est juste la somme des plaques qui fusionnent, mais il faut enregistrer cette information et la renvoyer dans le programme principal).
Pour le programme principal, la partie délicate est la mise en forme de la liste temporaire T, qui se substitue à la liste « officielle » K du contenu des cases. Je m'arrange pour que cette liste T contienne 4 colonnes à traiter toujours de la même façon, même si, dans liste K les lignes ou colonnes ne sont pas dans le sens du traitement unique réalisé dans move(). Je détecte qu'il y a eu un mouvement grâce à l'additif add au score qui vaut 0 si il y a un mouvement sans fusion de plaques, un nombre supérieur à 0 si il y a fusion de plaques et -1 si il n'y a pas de mouvement dans aucune des 4 colonnes.

```
from random import *
from math import *
from kandinsky import *
from utilitaires import *

def carre(x,y,c,l):
  for i in range(l):
    for j in range(l):
      set_pixel(x+i,y+j,c)

def plateau():
  carre(60,10,color(150,150,150),200)
  for i in range(16):
    carre(64+(i%4)*49,14+(i//4)*49,color(200,200,200),45)
  draw_string("score",265,20)
  affiche(0)

def affiche(s):
  draw_string(str(s),265,40)

def tuiles(k):
  col=[color(200,200,200),color(238,228,218),color(237,224,200),
  color(242,177,121),color(245,149,99),color(246,124,95),
  color(246,94,59),color(237,207,114),color(237,204,97),
  color(237,200,80),color(236,198,65),color(224,194,46),
  color(255,61,61),color(255,30,32)]#jusqu'a 8196
  for i in range(16):
    c=k[i%4][i//4]
    x,y=64+(i%4)*49,14+(i//4)*49
    carre(x,y,col[c],45)
    if c!=0:draw_string(str(2**c),x+22-5*len(str(2**c)),y+15)

def valeur():
  if randint(1,7)==1:return 2
  return 1

def entrer():
  c=int(input())
  if c==5 or c==2 or c==4 or c==6 or c==8 :return c
  else : return entrer()
```

```
def j(s):
  seed(s)
  score=0
  K=[4*[0],4*[0],4*[0],4*[0]]
  case=randint(0,15)
  K[case%4][case//4]=valeur()
  case+=randint(1,15)
  K[case%4][(case%16)//4]=valeur()
  plateau()
  tuiles(K)
  sens=entrer()
  while sens!=5:
    T=[4*[0],4*[0],4*[0],4*[0]]
    moved=False
    for i in range(4):
      for j in range(4):
        if sens==6:T[i][j]=K[j][i]
        if sens==4:T[i][j]=K[3-j][i]
        if sens==8:T[i][j]=K[i][3-j]
        if sens==2:T[i][j]=K[i][j]
      T[i],add=move(T[i])
      if add>=0:
        moved=True
        score+=add
    if moved==True:
      for i in range(4):
        for j in range(4):
          if sens==6:K[i][j]=T[j][i]
          if sens==4:K[i][j]=T[j][3-i]
          if sens==8:K[i][j]=T[i][3-j]
          if sens==2:K[i][j]=T[i][j]
      affiche(score)
      tuiles(K)
      case=randint(0,15)
      while K[case%4][case//4]!=0:
        case=randint(0,15)
      K[case%4][case//4]=valeur()
      tuiles(K)
    sens=entrer()
```

Le traitement qui est dans move() est donné ci-dessous. L'ensemble ne rentrant pas dans un seul fichier (4096 octets maximum), je l'ai mis dans un fichier à part. Il faut télécharger les deux fichiers dans la calculatrice et exécuter d'abord ce fichier *utilitaires.py*, ensuite exécuter *jeu2048.py* et enfin lancer la fonction j() avec sa graine, par exemple en tapant j(123).
Si on veut comprendre le fonctionnement du traitement move(), il serait bien que j'explique comment j'ai fait. J'ai distingué les différents cas qui peuvent se présenter et je les ai représentés par la notation suivante, appliquée aux 3 cas où on va avoir une fusion entre les deux premières plaques (celles qui sont collées au bord vers lequel on veut déplacer la ligne) :

```
if t[3]==t[2] and t[3]!=0:
  t[3]=1+t[2]
  if t[1]==t[0] and t[1]!=0:
    t[2]=1+t[1]
    t[1]=0
    moved+=2**t[2]
  elif t[1]==0:t[2]=t[0]
  else:#t[1]!=0
    t[2]=t[1]
    t[1]=t[0]
  moved+=2**t[3]
```

B B A A >>>> 0 0 2B 2A

B 0 A A >>>> 0 0 B 2A

B C A A >>>> 0 B C 2A

Selon le même procédé, j'ai fait la liste des autres mouvements possibles sur une ligne se déplaçant vers la droite. Il y a 13 autres cas, chacun a donc été transformé en instructions de programme pour construire la liste t qui est renvoyée au programme principal. Pour simplifier la lecture, j'ai mis des barres verticales qui

matérialisent les indentations (sinon elles sont difficiles à suivre).

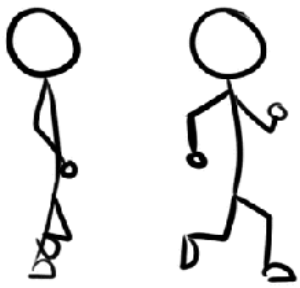
```
def move(t):
  moved=0
  if t[3]==t[2] and t[3]!=0:
    t[3]=1+t[2]
    if t[1]==t[0] and t[1]!=0:
      t[2]=1+t[1]
      t[1]=0
      moved+=2**t[2]
    elif t[1]==0:t[2]=t[0]
    else:#t[1]!=0
      t[2]=t[1]
      t[1]=t[0]
    moved+=2**t[3]
  elif t[3]!=0:
    if t[2]==t[1] and t[2]!=0:
      t[2]=1+t[2]
      t[1]=t[0]
      moved+=t[2]
    elif t[1]==0 and t[2]!=0:
      if t[2]==t[0]:
        t[2]=1+t[2]
        moved+=2**t[2]
      else:#t[2]!=t[0]
        t[1]=t[0]
    elif t[2]!=0 and t[1]==t[0] and t[1]!=0:
      t[1]=1+t[1]
      moved+=2**t[1]
    elif t[2]==0:
      if t[1]==t[3]:
        t[3]=1+t[3]
        t[2]=t[0]
        moved+=2**t[3]
      elif t[1]!=0 :
        if t[1]==t[0]:
          t[2]=1+t[1]
          t[1]=0
          moved+=2**t[2]
        else:#t[1]!=t[0]
          t[2]=t[1]
          t[1]=t[0]
      else:#t[1]==0
        if t[0]==t[3]:
          t[3]=1+t[3]
          moved+=2**t[3]
        else:#t[0]!=t[3]
          t[2]=t[0]
    else:moved=-1
  elif t[3]==0:
    if t[2]!=0 :
      if t[2]==t[1]:
        t[3]=1+t[2]
        t[2]=t[0]
        t[1]=0
        moved+=2**t[3]
      else:#t[2]!=t[1]
        if t[1]==t[0] and t[1]!=0 :
          t[3]=t[2]
          t[2]=1+t[1]
          t[1]=0
          moved+=2**t[1]
        elif t[1]!=t[0] and t[1]!=0:
          t[3]=t[2]
          t[2]=t[1]
          t[1]=t[0]
        elif t[1]==0:
          if t[0]==t[2]:
            t[3]=1+t[2]
            moved+=2**t[3]
          else:#t[0]!=t[2]
            t[3]=t[2]
            t[2]=t[0]
    else:#t[2]==0
      if t[1]==t[0] and t[1]!=0 :
        t[3]=1+t[1]
        t[1]=0
        moved+=2**t[3]
      elif t[1]!=t[0] and t[1]!=0 :
        t[3]=t[1]
        t[2]=t[0]
        t[1]=0
      elif t[0]!=0:
        t[3]=t[0]
  if moved>=0:t[0]=0
  return t,moved
```

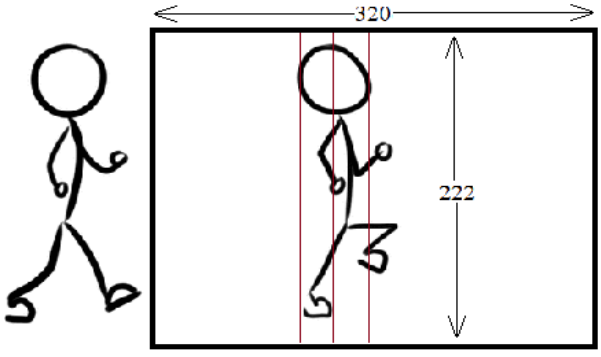
Voilà ! Il ne reste plus qu'à jouer pour tester le programme. J'ai fait un essai, mais je ne suis pas allé assez loin (je n'ai même pas encore atteint la plaque 1024... Si vous testez le programme et que vous trouvez des erreurs n'hésitez pas à me le signaler !

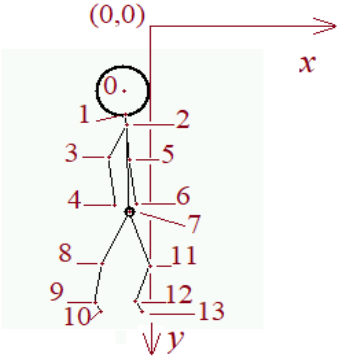
4. Personnage dansant

Ce jeu n'en est pas un ; il s'agit d'une animation. On voudrait dessiner un personnage très simple (une tête ronde, 2 yeux, une bouche, 2 segments pour les bras et les jambes...) et l'animer sur l'écran : le faire danser n'est pas forcément le plus simple. On peut se contenter de le faire se déplacer, éventuellement on peut imaginer commander son mouvement avec les touches de la calculatrice. Il n'y a donc pas de règle, on laisse carte blanche à la créativité de chacun.

Je vais essayer de faire marcher un personnage filiforme avec une tête et un bassin dessinés par un cercle. Il faut se limiter dans les ambitions car on a très peu de mémoire à disposition et les possibilités graphiques sont aussi très limitées puisqu'il n'y a qu'une seule instruction, set_pixel(). Une partie de la mémoire est déjà mobilisée par les deux fonctions qui définissent les tracé d'un segment ou d'un cercle. Il faut ensuite créer une autre fonction qui trace le personnage quand on connaît les coordonnées des points d'articulation de ses membres. Mon personnage est défini par la connaissance de 14 points.







Les coordonnées utiles d'une image statique peuvent ainsi être définie, par exemple par la liste suivante qui traduit en nombres l'image de droite de mon illustration (la partie de gauche vient d'un « gif animé » trouvé sur internet qui m'a servi de modèle pour dessiner mon personnage) : P=[[-20,37],[-18,57],[-17,63],[-30,90],[-25,125],[-15,90],[-10,125],[-15,130],[-35,171],[-40,199],[-35,208],[0,171],[-10,199],[-5,208]].
Avec ces quelques préliminaires, on peut commencer à penser à une animation de ces points pour que le personnage ne reste pas statique. La 1ère animation que j'ai réalisée consiste simplement en une translation régulière des points de manière à ce que le personnage défile horizontalement de la gauche vers la droite, entrant en scène d'un côté puis sortant de l'autre, la scène pouvant être rejouée plusieurs fois. Le programme qui réalise cela est donné ci-dessous, il contient déjà 1610 octets (la limite étant 4096 octets).

```
from kandinsky import *
from math import *
def cercle(x0,y0,r,c,e):
  for i in range(2*e):
    xd=x0-int((r-i*0.5)/sqrt(2))
    xf=x0+int((r-i*0.5)/sqrt(2))
    for x in range(xd,xf+1):
      x1=x
      y1=y0+int(sqrt((r-i*0.5)**2-(x-x0)**2))
      set_pixel(x,y1,c)
      for j in range(3):
        x2=x0+y1-y0
        y2=y0+x0-x1
        set_pixel(x2,y2,c)
        x1,y1=x2,y2

def seg(xa,ya,xb,yb,c):
  if abs(yb-ya)<abs(xb-xa):
    if xb<xa:
      xa,xb=xb,xa
      ya,yb=yb,ya
    m=(yb-ya)/(xb-xa)
    p=ya-m*xa
    for i in range(xb-xa):
      set_pixel(int(xa+i),
      int(m*(xa+i)+p),c)
  else:
    if yb<ya:
      ya,yb=yb,ya
      xa,xb=xb,xa
    m=(xb-xa)/(yb-ya)
    p=xa-m*ya
    for i in range(yb-ya):
      set_pixel(int(m*(ya+i)+p),
      int(ya+i),c)

def temporise(c):
  for i in range(500):
    c=(c**2)%12345

def personnage(p,c):
  gus=[[1,2],[2,3],[3,4],[2,5],
  [5,6],[2,7],[7,8],[8,9],[9,10],
  [7,11],[11,12],[12,13]]
  cercle(p[0][0],p[0][1],20,c,2)
  cercle(p[7][0],p[7][1],4,c,2)
  for i in range(len(gus)):
    seg(p[gus[i][0]][0],p[gus[i][0]][1],
    p[gus[i][1]][0],p[gus[i][1]][1],c)

def marche(n):
  for i in range(n):
    P=[[-20,37],[-18,57],[-17,63],#tete
    [-30,90],[-25,125],#bras gauche
    [-15,90],[-10,125],#bras droit
    [-15,130],#bassin
    [-35,171],[-40,199],[-35,208],#j.gau.
    [0,171],[-10,199],[-5,208]]#j.dr.
    t,M=0,P
    while t<80:
      personnage(M,color(255,255,255))
      M=[]
      for i in range(len(P)):
        M.append([P[i][0]+5*t,P[i][1]])
      personnage(M,color(0,0,0))
      t+=1
      temporise(98765)
```

Trois petites précisions sur cette version du programme :

- Avant de dessiner le personnage dans la position t, il faut effacer le personnage en position t-1. Une autre possibilité pour effacer ce personnage serait de passer tout le rectangle contenant le personnage en blanc, mais j'ai pensé que ce serait moins efficace (il aurait peut-être fallu tester ces deux options pour mieux les comparer).
- Je me suis senti obligé d'utiliser deux listes P et M car j'ai dans l'idée d'effectuer d'autres traitements que la simple translation qui aurait pu être réalisée plus simplement, juste en ajoutant régulièrement 5 pixels (c'est la valeur retenue ici) à P[i][o] qui est l'abscisse du point i, pour i allant de 0 à 13. Les autres traitements, on va le voir ci-dessous, modifient plus simplement la position initiale que la position à l'instant t-1.
- La fonction temporise() qui a été utilisée est la même que dans mon programme « horloge.py ». J'ai juste changé la valeur du range (j'ai mis 500 au lieu de 7800) pour régler la vitesse du défilement.

Pour animer les bras et les jambes d'un mouvement d'oscillation, on peut imaginer, dans un 1er temps que le bras est une entité solidaire (le coude ne permettant donc pas d'oscillation secondaire) qui oscille autour de son point d'ancrage, ici le point n°2. Pareil pour chaque bras et aussi pour les jambes qui oscilleraient autour du point n°7.

Concentrons-nous sur l'oscillation d'un seul bras :
il s'agit d'un mouvement pendulaire, une rotation autour du point d'ancrage, à la façon d'un pendule. Le bras étant initialement à un angle θ_0 de la verticale, cet angle va suivre une loi mathématique qui peut être modélisée par une fonction circulaire :
$\theta = \theta_0 \times \cos(\omega \times t)$ où le paramètre ω est appelé pulsation. Cette pulsation nous donne la période de la fonction $T = \frac{2\pi}{\omega}$, et donc, si on veut que la période soit de T=16, il faut prendre $\omega = \frac{2\pi}{T} = \frac{\pi}{8}$. Bon, ça c'est la théorie, passons à la pratique. Les coordonnées cartésiennes (x et y) d'un point situé à une distance R de l'origine (0,0) a les coordonnées suivantes : $x = R\sin\theta$ et $y = R\cos\theta$.
Calculons donc R1 et R2 pour les 2 points d'un bras ainsi que th1 et th2 les 2 angles initiaux de ce bras et effectuons la correction de ses coordonnées pour obtenir ce mouvement pendulaire. Le résultat est à peu près satisfaisant, le bras se balançant comme il se doit. Adaptons cette méthode à l'autre bras.

pour animer un bras seulement

```
def marche(n):
  P=[[-20,37],[-18,57],[-17,63],#tete
  [-30,90],[-25,125],#bras gauche
  [-15,90],[-10,125],#bras droit
  [-15,130],#bassin
  [-35,171],[-40,199],[-35,208],#j.gau.
  [0,171],[-10,199],[-5,208]]#j.dr.
  R1=sqrt((P[2][0]-P[3][0])**2+(P[2][1]-P[3][1])**2)
  R2=sqrt((P[2][0]-P[4][0])**2+(P[2][1]-P[4][1])**2)
  th1=atan((P[2][0]-P[3][0])/(P[2][1]-P[3][1]))
  th2=atan((P[2][0]-P[4][0])/(P[2][1]-P[4][1]))
  for i in range(n):
    t,M=0,P
    while t<80:
      personnage(M,color(255,255,255))
      M=[]
      for i in range(len(P)):
        M.append([P[i][0]+5*t,P[i][1]])
      M[3][0]=M[2][0]+int(R1*sin(th1*cos((pi*t)/8)))
      M[3][1]=M[2][1]+int(R1*cos(th1*cos((pi*t)/8)))
      M[4][0]=M[2][0]+int(R2*sin(th2*cos((pi*t)/8)))
      M[4][1]=M[2][1]+int(R2*cos(th2*cos((pi*t)/8)))
      personnage(M,color(0,0,0))
      t+=1
      temporise(98765)
```

```
from kandinsky import *
from math import *

def cercle(x0,y0,r,c):
  for i in range(2):
    xd=x0-int((r-i*0.5)/sqrt(2))
    xf=x0+int((r-i*0.5)/sqrt(2))
    for x in range(xd,xf+1):
      x1=x
      y1=y0+int(sqrt((r-i*0.5)**2-(x-x0)**2))
      set_pixel(x,y1,c)
      for j in range(3):
        x2=x0+y1-y0
        y2=y0+x0-x1
        set_pixel(x2,y2,c)
        x1,y1=x2,y2

def seg(xa,ya,xb,yb,c):
  if abs(yb-ya)<abs(xb-xa):
    if xb<xa:xa,xb,ya,yb=xb,xa,yb,ya
    m=(yb-ya)/(xb-xa)
    for i in range(xb-xa):
      set_pixel(xa+i,int(m*i+ya),c)
  else:
    if yb<ya:ya,yb,xa,xb=yb,ya,xb,xa
    m=(xb-xa)/(yb-ya)
    for i in range(yb-ya):
      set_pixel(int(m*i+xa),ya+i,c)

def personnage(p,c):
  gus=[[1,2],[2,3],[3,4],[2,5],
  [5,6],[2,7],[7,8],[8,9],[9,10],
  [7,11],[11,12],[12,13]]
  cercle(p[0][0],p[0][1],20,c)
  cercle(p[7][0],p[7][1],4,c)
  for i in range(len(gus)):
    seg(p[gus[i][0]][0],p[gus[i][0]][1],
    p[gus[i][1]][0],p[gus[i][1]][1],c)
```

```
def marche():
  P=[[-20,37],[-18,57],[-17,63],#tete
  [-30,90],[-25,125],#bras gauche
  [-15,90],[-10,125],#bras droit
  [-15,130],#bassin
  [-35,171],[-40,199],[-35,208],#j.gau.
  [0,171],[-10,199],[-5,208]]#j.dr.
  R1=sqrt((P[2][0]-P[3][0])**2+(P[2][1]-P[3][1])**2)
  R2=sqrt((P[2][0]-P[4][0])**2+(P[2][1]-P[4][1])**2)
  th1=atan((P[2][0]-P[3][0])/(P[2][1]-P[3][1]))
  th2=atan((P[2][0]-P[4][0])/(P[2][1]-P[4][1]))
  R3=sqrt((P[7][0]-P[8][0])**2+(P[7][1]-P[8][1])**2)
  R4=sqrt((P[7][0]-P[9][0])**2+(P[7][1]-P[9][1])**2)
  th3=atan((P[7][0]-P[8][0])/(P[7][1]-P[8][1]))
  c=999
  for t in range(80):
    personnage(P,color(255,255,255))
    for i in [0,1,2,7]:P[i][0]+=5
    t1=th1*cos((pi*t)/8)
    t2=th2*cos((pi*t)/8)
    t2b=-t2
    if sin(t1)>0: t2+=t1
    else: t2b-=t1
    t3=th3*cos((pi*t)/8)
    P[3][0]=P[2][0]+int(R1*sin(t1))
    P[3][1]=P[2][1]+int(R1*cos(t1))
    P[4][0]=P[2][0]+int(R2*sin(t2))
    P[4][1]=P[2][1]+int(R2*cos(t2))
    P[5][0]=P[2][0]+int(R1*sin(-t1))
    P[5][1]=P[2][1]+int(R1*cos(-t1))
    P[6][0]=P[2][0]+int(R2*sin(t2b))
    P[6][1]=P[2][1]+int(R2*cos(t2b))
    P[8][0]=P[7][0]+int(R3*sin(t3))
    P[8][1]=P[7][1]+int(R3*cos(t3))
    P[9][0]=P[7][0]+int(R4*sin(t3*2))
    P[9][1]=P[7][1]+int(R4*cos(t3*2))
    P[11][0]=P[7][0]+int(R3*sin(-t3))
    P[11][1]=P[7][1]+int(R3*cos(-t3))
    P[12][0]=P[7][0]+int(R4*sin(-t3*2))
    P[12][1]=P[7][1]+int(R4*cos(-t3*2))
    P[10][0]=P[9][0]+10
    P[10][1]=P[9][1]
    P[13][0]=P[12][0]+10
    P[13][1]=P[12][1]
    personnage(P,color(0,0,0))
    for i in range(999):c=(c**2)%123
```

J'ai trouvé un système qui permet d'articuler les deux parties du bras. Pour donner plus de ballant à l'avant-bras (aux points 4 et 6), j'augmente l'angle du point 4 de la quantité t1 (l'angle donné au point 3) ou je diminue l'angle du point 5 de la même quantité t1 : `if sin(t1)>0: t2+=t1`
Cela fonctionne assez bien et je m'apprête à mettre au point un mécanisme similaire pour les jambes (pour elles c'est la cuisse qui monte d'avantage quand elle avance, le mollet restant plutôt en retrait, éventuellement il faudrait aussi augmenter l'extension de la jambe vers l'arrière), mais je suis arrêté par la capacité de mémoire de la calculatrice : un triste message me disant à maintes reprises que l'allocation de mémoire dépasse la limite... J'essaie de restreindre un peu tout ce que je peux mais je ne parviens pas à un résultat très satisfaisant pour les jambes. Regardez le résultat et essayez de simplifier l'ensemble. Il faudrait enlever des variables mais je ne voudrais pas y consacrer tout mon temps libre. Si vous trouvez une solution plus satisfaisante, faites le moi savoir (ph.moutou@free.fr) !

Après une courte réflexion, je propose finalement le programme suivant qui a le mérite de respecter davantage l'anatomie du mouvement. On y perd dans le ballant de l'avant-bras et l'extension du mollet pourrait aussi être améliorée, mais bon, la portée du résultat n'est pas si grande. Même amélioré, le personnage restera filiforme avec une inexpressive tête ronde. Si on veut vraiment mieux sur le plan graphique, on aura intérêt à explorer d'autres modes de programmation... C'est un bon exercice cependant, et non dénué de réflexion sur des objets mathématiques (ici des angles), que d'essayer de faire mieux ou de proposer d'autres scénarios (personnage dansant ou marchant de face, etc.)

```
def marche(n):
  for k in range(n):
    P=[[-20,37],[-18,57],[-17,63],#tete
    [-30,90],[-25,125],#bras gauche
    [-15,90],[-10,125],#bras droit
    [-15,130],#bassin
    [-35,171],[-40,199],[-35,208],#j.gau.
    [0,171],[-10,199],[-5,208]]#j.dr.
    R1,R2,R3,R4=30.0,62.514,45.62,73.39
    th1,th2,th3,th4=-0.45,0.32,0.1,-0.1
    c=999
    for t in range(80):
      personnage(P,color(255,255,255))
      for i in [0,1,2,7]:P[i][0]+=5
      co=cos((pi*t)/8)
      P[3][0]=P[2][0]+int(R1*sin(th1*co))
      P[3][1]=P[2][1]+int(R1*cos(th1*co))
      P[4][0]=P[2][0]+int(R2*sin(th2+th1*co))
      P[4][1]=P[2][1]+int(R2*cos(th2+th1*co))
      P[5][0]=P[2][0]+int(R1*sin(-th1*co))
      P[5][1]=P[2][1]+int(R1*cos(-th1*co))
      P[6][0]=P[2][0]+int(R2*sin(th2-th1*co))
      P[6][1]=P[2][1]+int(R2*cos(th2-th1*co))
      P[8][0]=P[7][0]+int(R3*sin(th3+th1*co))
      P[8][1]=P[7][1]+int(R3*cos(th3+th1*co))
      P[9][0]=P[7][0]+int(R4*sin(th4+th1*co))
      P[9][1]=P[7][1]+int(R4*cos(th4+th1*co))
      P[11][0]=P[7][0]+int(R3*sin(th3-th1*co))
      P[11][1]=P[7][1]+int(R3*cos(th3-th1*co))
      P[12][0]=P[7][0]+int(R4*sin(th4-th1*co))
      P[12][1]=P[7][1]+int(R4*cos(th4-th1*co))
      P[10][0]=P[9][0]+10
      P[10][1]=P[9][1]
      P[13][0]=P[12][0]+10
      P[13][1]=P[12][1]
      personnage(P,color(0,0,0))
      for i in range(999):c=(c**2)%123
```