Sur la Construction d'une Animation

jpr, L1 Info & Math (Oct 2014)

Dans une scène de dimension donnée, vous devez bâtir une **animation** qui termine lorsqu'une certaine condition est réalisée. Le squelette est le suivant. Notez que j'encapsule immédiatement le tout dans une fonction, avec des constructions *locales* à cette fonction, c'est plus propre ! Plus tôt on prend les bonnes habitudes…

Le squelette du programme [cours 4]

```
(define (mon-animation)
  (local [(define LARG ...)          ; largeur de la scène
          (define HAUT ...)          ; hauteur de la scène
          (define FOND ...)          ; le fond invariant de la scène
          ; …… J'explique ici mon choix du modèle mathématique m du Monde
          (define INIT ...)          ; le Monde initial
          (define (suivant m)              ; l'évolution : Monde → Monde
            ...)
          (define (dessiner m)             ; la visualisation : Monde → Scène
            ...)
          (define (final? m)               ; l'apocalypse : Monde → Boolean
            ...)
          ; et toutes autres fonctions auxilliaires de mon animation
          ...]
    (big-bang INIT                         ; la Création
              (on-tick suivant)
              (on-draw dessiner)
              (stop-when final?))))
```

Soyez sensibles à *l'indentation* [distance à la marge], c'est elle qui montre la structure de votre programme. Dans le doute, utilisez Ctrl-I (Windows) ou Cmd-I (Mac) pour réaligner toutes les lignes du fichier. Une mauvaise indentation [encore faut-il la voir !] est alors une indication d'erreur de parenthèse. Mais vous ne faites plus d'erreurs de parenthèses, n'est-ce pas ?…

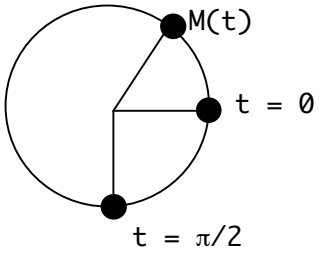
So, what is the Problem ?

Le problème essentiel réside dans le choix du **modèle mathématique du Monde**. A un instant donné, **quelles sont les variables qui gouvernent le Monde ?** Dans une première approche, nous avons *un Monde à 1 paramètre*, dont l'évolution est entièrement guidée par une seule variable. Prenons l'exemple d'une balle qui décrit un cercle centré au milieu de la scène et de rayon donné [exo TD 4.1]. A tout moment, la scène dépend uniquement de la position de la balle. De combien de variables ai-je besoin ? Une réponse naïve qui dirait : *j'ai besoin des coordonnées (x,y) de la balle*, ne verrait pas qu'elles sont liées entre elles [par l'équation du cercle]. Non. Comme les physiciens qui décrivent l'état d'un système, nous cherchons des variables indépendantes, ce que les physiciens nomment justement des *degrés de liberté*. Bref, ici, il suffit

de se souvenir que si M(x,y) est un point mobile sur un cercle de centre (a,b) et de rayon r, ses coordonnées sont données par les **équations paramétriques** :

```
x(t) = a + r cos(t)        y(t) = b + r sin(t)              (*)
```

Et la voilà, ma variable qui va décrire l'état du système, c'est l'angle `t`. Attention, à cause des orientations des axes en programmation, le sens *direct* est celui des aiguilles d'une montre !



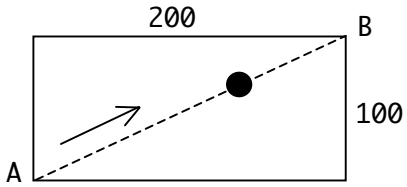
Mettons en forme l'animation de ce *Monde à 1 paramètre*. Je décide qu'à chaque top d'horloge [*tick*] mon angle `t` augmentera de 0.05 radians [environ 3 degrés]. Enfin, l'animation stoppera lorsque la balle aura fait 3 tours complets :

```
(define (balle-qui-tourne)
  (local [(define TAILLE 100)           ; la dimension de la scène carrée
          (define TAILLE/2 (quotient TAILLE 2))      ; pour ne pas le recalculer !
          (define R 35)                 ; le rayon de la trajectoire circulaire
          (define BALLE (circle 10  'solid "red"))          ; l'image de la balle
          (define FOND (rectangle TAILLE TAILLE 'solid "yellow"))
          ; le Monde mathématique sera l'angle polaire t de la balle
          (define INIT 0)               ; le Monde initial (pôle Est du cercle)
          (define (suivant t)           ; Monde → Monde
            (+ t 0.05))                 ; le choix de 0.05 est empirique
          (define (dessiner t)          ; Monde → Scène
            (place-image BALLE
                         (+ TAILLE/2 (* R (cos t)))
                         (+ TAILLE/2 (* R (sin t)))
                         FOND))
          (define (final? t)   ; Monde → Boolean
            (>= t (* 6 pi)))]
    (big-bang INIT
              (on-tick suivant 1/50)    ; horloge au 1/50ème de sec
              (on-draw dessiner)
              (stop-when final?))))
```

Vu ? **Le problème essentiel est le choix de la variable mathématique qui gouverne le Monde**. Notez au passage que les équations (*) ne sont pas uniques, elles décrivent un mouvement circulaire à vitesse constante ! Essayez donc un mouvement accéléré…

Un autre exemple : mouvement rectiligne

Prenons l'exemple d'une balle qui doit traverser une scène rectangulaire 200 x 100, du point A au point B, pour stopper en B.



L'état de la scène est parfaitement déterminé par l'abcisse `x` de la balle, dont la trajectoire se situe sur la droite AB d'équation `y = ax + b`. Si on impose à la droite de passer par les points A et B, cela donne les équations `100 = b` et `0 = 200a + b`, système qui se résoud en `a = -1/2` et `b = 100`. L'équation de la trajectoire est donc `y = - x/2 + 100`, avec `x ∈ [0,200]`.

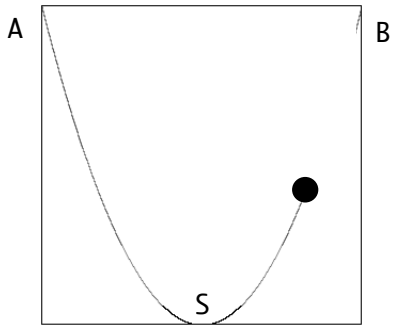
L'avancée de la balle est déterminée par la croissance de son abcisse `x`. Le plus simple consiste à faire avancer `x` d'un pixel à chaque top d'horloge. En jouant sur la fréquence, ici laissée à 1/28 sec par défaut, on pourra régler la vitesse de l'animation.

```
(define (balle-sur-diagonale)
  (local [(define FOND (rectangle 200 100 'solid "yellow"))
          (define BALLE (circle 10 'solid "red"))
          ; Le Monde est l'abcisse x de la balle, dans [0,200]
          (define INIT 0)
          (define (f x)
            (- 100 (/ x 2)))           ; équation de la trajectoire
          (define (suivant x)
            (+ x 1))
          (define (dessiner x)
            (place-image BALLE x (f x) FOND))
          (define (final? x)
            (>= x 200))]
    (big-bang INIT
              (on-tick suivant)
              (on-draw dessiner)
              (stop-when final?)
              (state true))))          ; pour voir l'état du monde !
```

La dernière ligne **(state true)** est une spécification du `big-bang` que nous n'avons pas vue en cours. Il s'agit d'un outil de mise au point ouvrant une fenêtre supplémentaire affichant l'état du Monde [donc ici de `x`] au fur et à mesure de l'avancée de l'animation. Cela permet de détecter des erreurs…

Pourquoi se limiter à une seule variable ?

En effet, à partir du moment où l'on dispose des **structures**, analogue des n-uplets en maths, on peut modéliser des systèmes à plusieurs degrés de liberté, en regroupant les variables indépendantes dans une *struct*. Prenons cette fois l'animation d'une balle qui se déplace sur une *parabole* passant par A et B et de sommet S, dans une scène carrée de côté 200. La balle part du point A et stoppe au point B. Il s'agit de l'exercice 5.6.4 du livre de cours PCPS :



Mais je veux en plus conserver la trace de la trajectoire !!

L'équation de la parabole est `y = f(x) =` ax^2+bx+c. Si l'on met en équation les conditions `f(0) = 0`, `f(300) = 0`, `f(150) = 300`, il vient `f(x) =` $-x^2/75 + 4x$, d'accord ? Reste à déterminer les variables du système. D'une part la position de la balle est connue dès qu'on connaît son abscisse `x`. D'autre part, si l'on veut conserver la trace du mouvement de la balle, il faudra traîner dans le monde une image que l'on va augmenter chaque fois d'un tout petit segment [la courbe étant vue comme une suite de très petits segments]. J'opte donc pour une *struct* à deux champs `(img,x)`.

```
(define (balle-parabolique)
  (local [(define TAILLE 300)
          (define FOND (rectangle TAILLE TAILLE 'solid "yellow"))
          (define BALLE (circle 20 'solid "red"))
          ; le monde m est un couple (img, x) ou x est l'abscisse de la balle
          (define-struct monde (img x))
          (define INIT (make-monde FOND 0))
          (define (f x)
            (+ (/ (* x x) -75) (* 4 x)))
          (define (suivant m)
            (local [(define img (monde-img m)) (define x (monde-x m))]
              (make-monde (add-line img x (f x) (+ x 3) (f (+ x 3)) "black")
                          (+ x 3))))   ; en modifiant 3 on joue sur la vitesse
          (define (dessiner m)
            (local [(define img (monde-img m)) (define x (monde-x m))]
              (place-image BALLE x (f x) img)))
          (define (final? m)
            (>= (monde-x m) TAILLE))]
    (big-bang INIT
              (on-tick suivant 1/60)
              (on-draw dessiner)
              (stop-when final?))))
```

Reprenons l'exemple du mouvement rectiligne

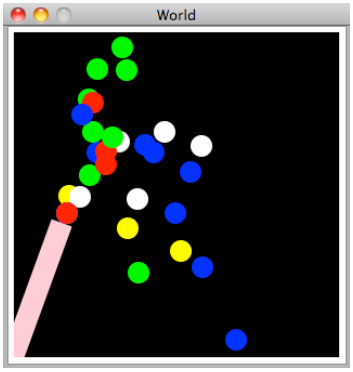
Reprenons l'exemple de la page 3, et modifions-le pour que la balle fasse des aller-retours entre A et B. Connaître uniquement l'abcisse `x` de la balle ne suffit plus, il faut aussi savoir si elle va de A vers B ou de B vers A. Introduisons à cet effet une seconde variable d'état `dir` qui représentera la direction, et qui vaudra –1 ou +1, quantité qui sera rajoutée à `x` à chaque top d'horloge.
Du coup le Monde devient le couple `(x,dir)` et nous optons pour une structure à deux champs !

```
(define (anim-diag-rebond)
  (local [(define FOND (rectangle 200 100 'solid "yellow"))
          (define BALLE (circle 10 'solid "red"))
          ; Le Monde est le couple (x,dir) avec x dans [0,200] et dir = +-1
          (define-struct monde (x dir))
          (define INIT (make-monde 0 1))        ; on part du point A
          (define (f x)
            (- 100 (/ x 2)))
          (define (suivant m)     ; m est une struct, le resultat aussi !
            (local [(define x (monde-x m)) (define dir (monde-dir m))]
              (cond ((< x 0) (make-monde 0 1))
                    ((> x 200) (make-monde 200 -1))
                    (else (make-monde (+ x dir) dir)))))
          (define (dessiner m)   ; m est une struct, résultat une scène
            (local [(define x (monde-x m))]
              (place-image BALLE x (f x) FOND)))]
    (big-bang INIT
              (on-tick suivant 1/50)
              (on-draw dessiner)
              (state true))))     ; particulièrement intéressant ici…
```

J'ai éliminé la fin du Monde. Si l'on voulait par exemple stopper l'animation au bout de trois aller-retours, comment faire ? Facile, on prend une troisième variable `n` qui compte le nombre d'aller-retours, et du coup le Monde devient un triplet `(x,dir,n)`. Et zou !…

Utilisation des LISTES dans une animation [cours 6] : des particules !

La structure de **liste** est très flexible pour représenter un ensemble d'objets. Prenons l'exemple d'un canon qui crache des billes de couleur soumises à la gravitation. Pour avoir l'illusion d'un jet continu, on adopte la stratégie suivante : chaque fois qu'une bille s'enfonce dans le sol, elle va renaître au bout du canon avec une vitesse légèrement aléatoire !



Le *Monde mathématique* sera donc une **liste de billes**. Une bille sera représentée par une structure `#(struct:bille x y dx dy img)`. En effet, inutile de reconstruire à chaque image de

l'animation le disque représentant la bille. Elle garde la même image de sa naissance au bout du canon jusqu'à sa mort dans le sol…

```
(define (canon-à-billes)
  (local [(define SIZE 300)
          (define SIZE/2 (quotient SIZE 2))
          (define FOND
            (place-image (rotate -20 (rectangle 20 200 'solid "pink"))
                         10 270
                         (rectangle SIZE SIZE 'solid "black")))
          (define-struct bille (x y dx dy img))       ; structure d'une bille
          (define (random-bille)
            (make-bille 59 187  ; l'extrémité du canon
                        (+ 2 (random 5)) (- -5 (random 13))
                        (circle 10 'solid (case (random 5)
                                            ((0) "white")
                                            ((1) "blue")
                                            (else "red")))))
          ; Le Monde est une liste de 30 billes
          (define INIT (build-list 30 (lambda (i) (random-bille))))
          ; puisque j'applique une fonction sur tous les éléments de L, j'opte
          ; pour un 'map' (cours 9) plutot que programmer une récurrence...
          (define (suivant L)                              ; Monde → Monde
            (map (lambda (b)   ; pour chaque bille b de L
                   (if (> (bille-y b) SIZE)
                       (random-bille)
                       (make-bille (+ (bille-x b) (bille-dx b))
                                   (+ (bille-y b) (bille-dy b))
                                   (bille-dx b)
                                   (+ (bille-dy b) 1)
                                   (bille-img b))))
                 L))
          ; cette fois je procède par recurrence sur L
          (define (dessiner L)                             ; Monde → Scène
            (if (empty? L)
                FOND
                (local [(define b (first L))]
                  (place-image (bille-img b)
                               (- (bille-x b) 10) (- (bille-y b) 20)
                               (dessiner (rest L))))))]
    (big-bang INIT
              (on-tick suivant)
              (on-draw dessine))))
```

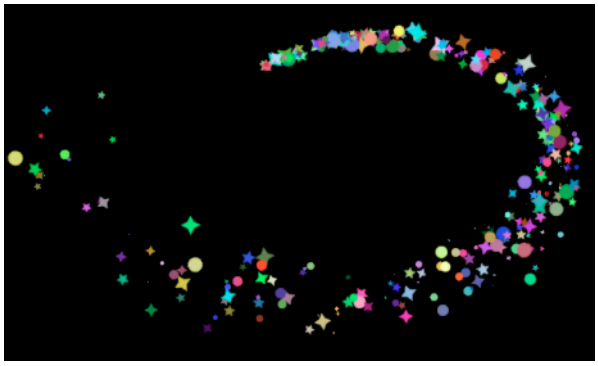
N.B. Si vous ne connaissez pas encore `map` (cours 9), vous programmerez la fonction `suivant` sous la forme d'une récurrence classique :

```
(define (suivant L)
  (if (empty? L)
      L
      (cons (if (> (bille-y b) SIZE
                   (random-bille)
                   (make-bille (+ (bille-x b) (bille-dx b))
                               (+ (bille-y b) (bille-dy b))
                               (bille-dx b)
                               (+ (bille-dy b) 1)
                               (bille-img b)))
            (suivant (rest L)))))
```

Techniquement, l'animation que nous venons de programmer se nomme un **système de particules**. Dans un tel système, toutes les particules sont des objets ayant le même comportement : chaque particule connaît sa position et sa vitesse, ainsi que la manière dont elle va se mouvoir à chaque top d'horloge. Elle peut aussi avoir la connaissance de sa durée de vie, etc. Les particules ne sont pas forcément des balles, ce peut être des aiguilles, des images quelconques, un brouillard, ou toute autre chose bizarre sortant de votre imagination :-)
La wikipedia vous en dira plus :

http://fr.wikipedia.org/wiki/Systeme_de_particules

Les logiciels professionnels de vidéo comme Motion ou Blender ont des dispositifs permettant de créer automatiquement des systèmes de particules. Mais si vous les créez vous-mêmes comme programmeur, peut-être vous faudra-t-il réviser un peu de physique ?



Comment programmer une animation à plusieurs étapes ?

Un étudiant m'a demandé un jour comment faire si l'animation ne faisait pas la même chose à chaque top d'horloge, mais procédait par *étapes*. On fait telle chose dans la première étape, puis autre chose dans la seconde, etc. En fait, il suffit de mettre le numéro de l'étape dans la structure du monde, et de se demander à quelle étape l'on est. Le cas échéant, on change d'étape… C'est le principe d'un **automate**. Prenons l'exemple suivant. Je veux dessiner des disques vides empilés à la verticale, puis ensuite seulement les remplir par des nombres aléatoires. Cela donnerait quelque chose comme :

```
(define (anim-etapes NB-JETONS)    ; ex : (anim-etapes 6)
 (local [(define RAYON 30)
         (define JETON (circle RAYON 'solid "blue"))
         (define SCENE (rectangle (* 2 RAYON) (* 2 RAYON NB-JETONS)
                                  'solid "white"))
         ; dans le monde, n = nombre de coups restants pour cette etape
         (define-struct monde (etape img n))
         (define INIT (make-monde 1 SCENE NB-JETONS))
         (define (suivant m)
           (local [(define e (monde-etape m))
                   (define img (monde-img m))
                   (define n (monde-n m))]
             (cond
               ((= e 1)
                (if (= n 1)                 ; fin etape 1 ?
                    (make-monde (+ e 1)     ; changement d'étape !
                                (place-image JETON RAYON RAYON img)
                                NB-JETONS)
                    (make-monde e           ; on reste dans l'étape 1
                                (place-image JETON RAYON
                                             (* (- (* 2 n) 1) RAYON) img)
                                (- n 1))))
               ((= e 2)
                (if (= n 0)                         ; fin etape 2 ?
                    (make-monde (+ e 1) img 0)      ; changement d'étape !
                    (make-monde e                   ; on reste en étape 2
                            (place-image
                               (text (number->string (random 100)) 24 "red")
                               RAYON (* (- (* 2 n) 1) RAYON)
                               img)
                            (- n 1))))
               (else m))))                          ; étape finale ☺
         (define (dessiner m)
           (monde-img m))
         (define (final? m)
           (= (monde-etape m) 3))]
   (big-bang INIT
             (on-tick suivant 1/4)              ; 4 images/sec
             (on-draw dessiner)
             (stop-when final?))))
```