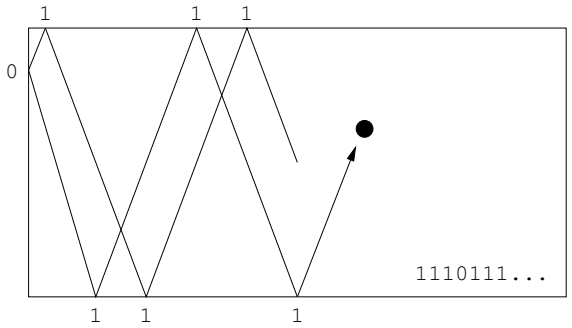
# TP2 : Mots du Billard

Considérons une boule se déplaçant sans perte d'énergie sur un billard. Si l'on note `0` à chaque fois que la boule touche un petit coté et `1` lorsqu'elle touche un grand coté, on peut décrire la suite des rebonds de la boule comme une suite infinie de `0` et de `1`. Une telle suite définit un *mot du billard*.




Le but du TP est de programmer une petite application graphique qui génère des mots du billard. Nous allons représenter la boule se déplaçant sur le billard et à chaque fois qu'elle touchera une bande, l'élément suivant de la suite sera affiché à l'écran.

Récupérer le fichier `billard.ml` à l'adresse `http://www.lri.fr/~mandel/aaf/tp-02`. Le compiler avec la ligne de commande suivante :

```
ocamlc -o billard graphics.cma billard.ml
```

Le fichier `billard.ml` définit les constantes `longueur`, `largeur`, `vx_init` et `vy_init` qui sont toutes de type `float`. La valeur de ces constantes peut être modifiée avec des arguments sur la ligne de commande (taper `./billard -help` pour plus d'informations).

Ce fichier contient également une fonction `affiche_boule: float -> float -> unit` qui à partir des positions d'une boule, l'affiche à l'écran.

**Question 1**
*Faites un programme qui affiche une boule au centre du billard.*
*Vous pouvez faire un appel à la fonction* `read_line: unit -> string` *pour attendre que l'utilisateur appuie sur la touche* `return` *avant que le programme termine.*

Pour simuler le déplacement de la boule de billard, le temps va être décomposé en une succession d'instants discrets. À chaque instant, à partir de la position de la boule à l'instant précédent et de sa vitesse, on calcule sa nouvelle position.

**Question 2**
*Écrire une fonction* `deplace_boule pre_x pre_y vx vy` *qui calcule la nouvelle position de la boule après une étape de calcul. La nouvelle position de la boule est obtenue en translatant le point (*`pre_x`, `pre_y`*) par le vecteur (*`pre_vx`, `pre_vy`*).*
*La fonction* `deplace_boule` *doit avoir le type suivant :*

```
val deplace_boule: float -> float -> float -> float -> float * float
```

**Question 3**
*Écrire une fonction récursive terminale* `simulation pre_x pre_y n` *qui simule le déplacement de la boule pendant* `n` *instants. À chaque instant, il faut calculer la nouvelle position de la boule et l'afficher. La fonction* `simulation` *doit avoir le type suivant :*

```
val simulation: float -> float -> int -> unit
```

**Question 4**
*Tester la fonction* `simulation`.

Pour réduire la vitesse de simulation, nous allons définir une fonction `attente n` qui s'appelle récursivement `n` fois puis retourne la valeur `()`.

**Question 5**
*Écrire le code de la fonction* `attente: int -> unit` *et modifier la fonction* `simulation` *pour ajouter un appel à la fonction* `attente` *à chaque étape de calcul.*

Nous allons maintenant gérer les rebonds en modifiant la vitesse lorsque la boule a dépassé l'une des bandes.

**Question 6**
*Écrire une fonction* `calcule_vitesse` *qui calcule la vitesse de la boule à chaque instant :*
- *si la boule est dans le cadre, la vitesse est inchangée ;*
- *si la boule a dépassé les limites par la gauche ou par la droite, la composante horizontale de la vitesse est inversée ;*
- *si la boule a dépassé les limites par le haut ou par le bas, la composante verticale de la vitesse est inversée.*

*La fonction* `calcule_vitesse` *doit avoir le type suivant :*

```
val calcule_vitesse: float -> float -> float -> float -> float * float
```

**Question 7**
*Modifier la fonction* `simulation` *pour qu'elle prenne les composantes horizontale et verticale de la vitesse précédente en paramètre. Ajouter la mise à jour de la vitesse avant le calcul de la nouvelle position.*

Nous voulons maintenant afficher les mots du billard. Pour cela, nous allons définir une fonctions `affiche_element` qui affiche `0` lorsque la boule touche une petite bande ou `1` lorsqu'elle touche une grande bande (le programme s'arrêtera sur une erreur si la boule touche deux bandes au même instant).

**Question 8**
*Écrire le code de la fonction* `affiche_element pre_vx pre_vy vx vy`. *Pour détecter les collisions, on peut simplement regarder si l'une des composantes du vecteur vitesse (*`vx`, `vy`*) a été inversée par rapport à la composante correspondante du vecteur vitesse de l'instant précédent (*`pre_vx`, `pre_vy`*). La fonction* `affiche_element` *doit avoir le type suivant :*

```
val affiche_element: float -> float -> float -> float -> unit
```

*Modifier la fonction* `simulation` *pour afficher les éléments du mot du billard généré.*

**Question 9**
*Modifier votre programme pour conserver dans une liste les éléments du mot du billard. Vous pourrez accumuler les éléments dans l'ordre inverse de leur arrivée puis renverser la liste.*

**Pour aller plus loin . . .**

Les mots du billard sont équilibrés, c'est-à-dire que la différence entre le nombre de `1` de deux sous mots de même taille est toujours inférieure ou égale à un. De plus, si l'angle initial est un nombre rationnel alors le mot est périodique, c'est-à-dire qu'il est constitué d'un motif qui se répète à l'infini. Plus d'informations sur :

`http://www.yann-ollivier.org/maths/sturm`