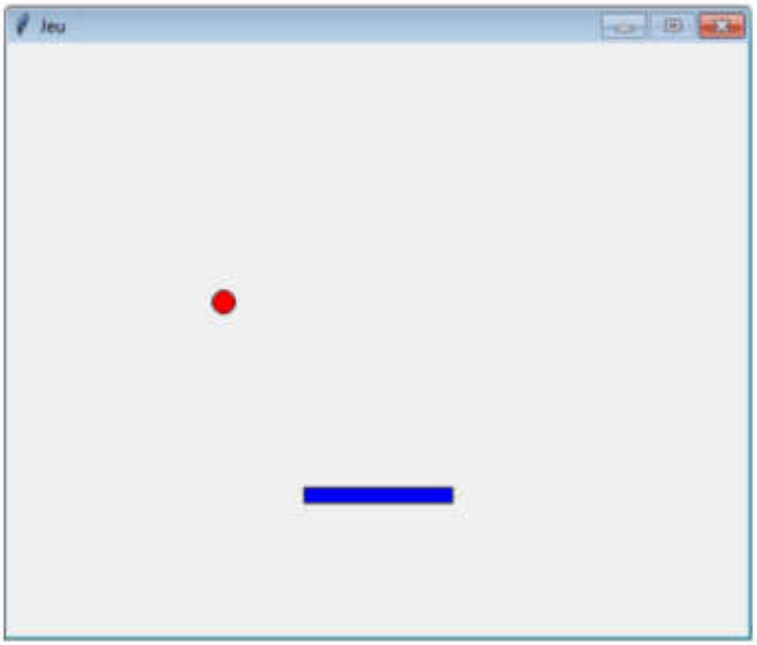
13

DÉBUTER TON PREMIER JEU : REBONDIR !

Jusqu'ici, nous avons vu les bases de la programmation. Tu as appris à utiliser des variables pour stocker des informations, les instructions `if` pour établir des conditions et les boucles `for` pour répéter du code. Tu sais désormais comment créer des fonctions pour réutiliser du code et exploiter des classes et des objets pour découper ton code en de petits extraits plus faciles à comprendre. Tu as appris aussi à dessiner des graphismes à l'écran avec les modules `turtle` et `tkinter`. Il est temps, à présent, de tirer parti de toutes ces connaissances pour créer ton premier jeu.

Frapper la balle magique

Nous allons développer un jeu avec une balle magique et une raquette. La balle vole à l'écran et le joueur doit la faire rebondir sur la raquette. Si, en retombant, la balle dépasse la raquette et touche le bas de l'écran, la partie est terminée. Voici un aperçu du jeu achevé :



Ce jeu peut paraître un peu simple, mais le code renferme quelques astuces et se distingue de tout ce que nous avons réalisé jusqu'ici ; il y a en effet de nombreuses choses à gérer. Ainsi, il faut animer la raquette et la balle, puis détecter si celle-ci touche les murs ou la raquette.

Dans ce chapitre, nous entamons la création du jeu, avec la génération du **canevas** et de la balle magique. Au chapitre suivant, nous le terminerons en lui ajoutant la raquette.

Créer le canevas du jeu

Pour créer ce jeu, commence par ouvrir une nouvelle fenêtre IDLE au départ du shell Python (menu `File>New File`). Dans cette fenêtre, importe `tkinter` et crée un **canevas** sur lequel tu pourras dessiner :

```
from tkinter import *
import random
import time
tk = Tk()
```

```
tk.title("Jeu")
tk.resizable(0, 0)
tk.wm_attributes("-topmost", 1)
canvas = Canvas(tk, width=500, height=400, bd=0, highlightthickness=0)
canvas.pack()
tk.update()
```

Ceci diffère quelque peu des exemples précédents. Nous importons tout d'abord les modules `random` et `time` pour pouvoir les utiliser par la suite dans le code.

La ligne `tk.title("Jeu")` se sert de la fonction `title` de l'objet `tk` créé avec `tk = Tk()` pour donner un titre à la fenêtre du canevas. La ligne suivante fait appel à la fonction `resizable` pour imposer une taille fixe à la fenêtre : les paramètres `0, 0` indiquent que la taille de la fenêtre ne peut être modifiée ni horizontalement, ni verticalement. Ensuite, nous appelons `wm_attributes` pour préciser à `tkinter` de placer la fenêtre du canevas au-dessus de toutes les autres fenêtres, c'est-à-dire à l'avant-plan (`"-topmost"`).

Note que, lors de la création de l'objet canevas avec `canvas =`, nous utilisons des paramètres supplémentaires par rapport aux exemples précédents. Ainsi, `bd=0` et `highlightthickness=0` garantissent qu'il n'y aura aucune bordure autour du canevas, ce qui donne à notre fenêtre un aspect plus adapté à notre écran de jeu.

La ligne `canvas.pack()` oblige le canevas à se redimensionner en fonction des paramètres de largeur et de hauteur fournis à la ligne précédente. Enfin, `tk.update()` indique à `tkinter` de s'initialiser lui-même en vue de l'animation du jeu. Sans cette dernière ligne, rien ne fonctionnerait comme prévu.

Enregistre le code à mesure que tu avances, avec un nom de fichier explicite, par exemple `rebondir.py`.

Créer la classe de la balle

Passons à la création de la classe de la balle. Pour débuter le code, il faut que celle-ci se dessine d'elle-même sur le canevas. Voici les étapes à suivre.

1. Crée une classe appelée `Balle`, qui prend des paramètres pour le canevas et la couleur de la balle à dessiner.

2. Enregistre le canevas dans une variable d'objet, parce que nous devons dessiner la balle dessus.
3. Trace un cercle plein sur le canevas à l'aide de la valeur du paramètre de couleur en tant que couleur de remplissage.
4. Mémorise l'identifiant que renvoie `tkinter` lorsqu'il dessine le cercle (un ovale, en fait), parce que nous en avons besoin pour déplacer la balle à l'écran.
5. Déplace l'ovale au milieu du canevas.

Ce code vient s'ajouter directement après les trois premières lignes du fichier (juste après `import time`).

```
from tkinter import *
import random
import time

❶ class Balle:
❷     def __init__(self, canvas, couleur):
❸         self.canvas = canvas
❹         self.id = canvas.create_oval(10, 10, 25, 25, fill=couleur)
❺         self.canvas.move(self.id, 245, 100)

❻     def dessiner(self):
          pass
```

Nous nommons notre classe `Balle` (❶) et créons ensuite (❷) la fonction d'initialisation (comme expliqué au chapitre 8), qui prend les paramètres `canvas` et `couleur`. Puis nous définissons la variable d'objet `canvas` avec la valeur du paramètre `canvas` (❸). Nous appelons la fonction `create_oval` avec cinq paramètres (❹) : les coordonnées *x* et *y* du coin supérieur gauche (`10, 10`), les coordonnées *x* et *y* du coin inférieur droit (`25, 25`) et, enfin, la couleur de remplissage de l'ovale.

La fonction `create_oval` renvoie un identifiant de la forme dessinée, que nous mémorisons dans la variable d'objet `id`. Ensuite, nous déplaçons (❺) l'ovale vers le milieu du canevas (`245, 100`). Ce dernier sait quel objet déplacer, parce que nous lui indiquons l'identifiant de la forme (la variable d'objet `id`).

Les deux dernières lignes de la classe `Balle` créent une fonction `dessiner(self)` (❻), dont le corps ne contient encore que le mot-clé `pass`. Pour l'instant, cette fonction ne fait encore rien, mais nous allons bientôt la compléter.

À ce stade, la classe `Balle` est définie et nous devons en créer une instance, un objet. Pour rappel, une classe décrit ce qu'elle est capable de réaliser, mais c'est l'objet qui le fait réellement. Ajoute le code suivant au bas du programme pour créer l'objet `balle` rouge :

```
balle = Balle(canvas, 'red')
```

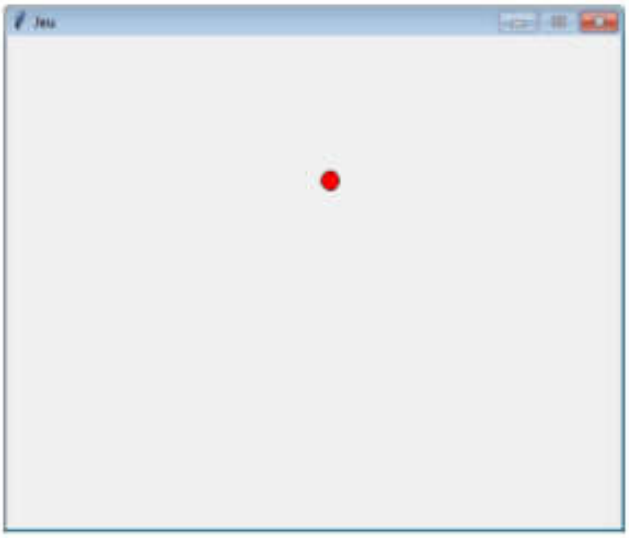
La chaîne `'red'` force la couleur de remplissage rouge (*red* en anglais). Si tu exécutes ce programme dans certaines versions antérieures de Python, le canevas apparaît pendant une fraction de seconde, puis disparaît. Pour empêcher la fenêtre de se fermer immédiatement, il faut ajouter une boucle d'animation appelée « boucle principale » de notre jeu. Même si la fenêtre ne se ferme pas de suite, nous avons besoin de cette boucle pour la suite des opérations.

La boucle principale constitue la partie centrale d'un programme et elle contrôle généralement l'essentiel de ce qu'il fait. Pour l'instant, elle s'exécute à l'infini (tant que tu ne fermes pas la fenêtre) en indiquant en permanence à `tkinter` de redessiner l'écran, puis en disant au programme de se mettre en sommeil pendant un centième de seconde.

Ajoute le code suivant à la fin du programme :

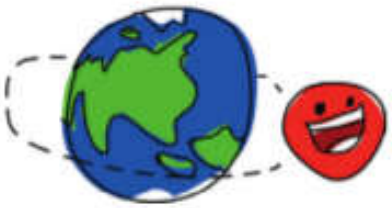
```
balle = Balle(canvas, 'red')

while 1:
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

Maintenant, quand tu exécutes le programme, la balle apparaît près du centre de l'écran. Remarque que la barre de titre contient `Jeu` et non plus `tk` :



Ajouter de l'action

À ce stade, la classe `Balle` est définie et nous avons un objet `balle`. Il nous reste à animer cette dernière, en la faisant avancer, rebondir et changer de direction.



Déplacer la balle

Pour déplacer la balle, il faut agir dans la fonction `dessiner`, comme suit :

```
class Balle:
    def __init__(self, canvas, couleur):
        self.canvas = canvas
        self.id = canvas.create_oval(10, 10, 25, 25, fill=couleur)
        self.canvas.move(self.id, 245, 100)

    def dessiner(self):
        self.canvas.move(self.id, 0, -1)
```

Comme `__init__` a mémorisé le paramètre `canvas` dans la variable d'objet du même nom, nous pouvons faire référence à cette variable sous la forme `self.canvas` et appeler la fonction `move` indirectement.

Nous passons trois paramètres à `move` : l'`id` de l'ovale ainsi que les nombres `0` et `-1`. Le `0` indique de ne pas bouger horizontalement, tandis que le `-1` qu'il faut se déplacer verticalement d'un pixel vers le haut de l'écran.

Cette petite modification est intéressante, car elle nous permet d'essayer des choses au fur et à mesure. Si nous devions écrire le code complet du jeu en une seule fois, que se passerait-il si quelque chose ne devait pas fonctionner comme prévu ? Il nous serait beaucoup plus difficile de trouver d'où vient le problème. En avançant comme ceci, petit à petit, il nous sera beaucoup plus facile ensuite de découvrir ce qui provoque des soucis et pourquoi.

L'autre modification vient se placer dans la boucle principale du bas du programme. Dans le bloc de la boucle `while` de notre boucle principale, nous ajoutons un appel à la fonction `dessiner` de l'objet `balle` :

```
while 1:
    balle.dessiner()
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

Maintenant, quand tu exécutes le programme, la balle avance vers le haut de l'écran pour disparaître, parce que le code oblige `tkinter` à redessiner l'écran très rapidement : les fonctions `update_idletasks` et `update` forcent `tkinter` à se dépêcher pour redessiner le contenu du canevas.

La fonction `sleep` (à savoir, « dormir ») du module `time` demande à Python de placer le programme en sommeil pendant un centième de seconde (`0.01`). Si tu ne mets pas une telle ligne, Python et `tkinter` vont tellement vite que tu n'as pas le temps de voir la balle qu'elle a déjà disparu de l'écran !

Fondamentalement donc, la boucle dit « déplacer la balle d'un peu, redessiner l'écran avec le nouvel emplacement, hiberner un instant, puis recommencer ».

NOTE *Lorsque le programme s'exécute et que tu fermes la fenêtre, le shell Python affiche un message d'erreur. Ceci est dû au fait que, quand tu fermes la fenêtre, le code est interrompu alors qu'il est dans la boucle principale et Python s'en plaint.*

Jusque-là, le code a l'allure suivante :

```
from tkinter import *
import random
import time

class Balle:
    def __init__(self, canvas, couleur):
        self.canvas = canvas
        self.id = canvas.create_oval(10, 10, 25, 25, fill=couleur)
        self.canvas.move(self.id, 245, 100)

    def dessiner(self):
        self.canvas.move(self.id, 0, -1)

tk = Tk()
tk.title("Jeu")
tk.resizable(0, 0)
tk.wm_attributes("-topmost", 1)
canvas = Canvas(tk, width=500, height=400, bd=0, highlightthickness=0)
canvas.pack()
tk.update()

balle = Balle(canvas, 'red')

while 1:
    balle.dessiner()
```

```
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

Faire rebondir la balle

Une balle qui disparaît en haut de l'écran ne présente pas vraiment d'intérêt pour un jeu ; ce serait mieux si elle rebondissait sur le mur. Pour cela, nous ajoutons quelques variables d'objet à la fonction d'initialisation de la classe `Balle`, comme suit :

```
class Balle:
    def __init__(self, canvas, couleur):
        self.canvas = canvas
        self.id = canvas.create_oval(10, 10, 25, 25, fill=couleur)
        self.canvas.move(self.id, 245, 100)
        self.x = 0
        self.y = -1
        self.hauteur_canevas = self.canvas.winfo_height()
```

Nous avons ajouté trois lignes supplémentaires au programme. Avec `self.x = 0`, nous définissons une variable d'objet `x` avec la valeur `0`, puis, avec `self.y = -1`, nous définissons la variable d'objet `y` avec la valeur initiale `-1`. Enfin, nous définissons la variable d'objet `hauteur_canevas` avec, comme valeur, le résultat de l'appel de la fonction `winfo_height` (qui renvoie la hauteur actuelle du canevas).

Ensuite, nous modifions de nouveau la fonction `dessiner` :

```
    def dessiner(self):
❶       self.canvas.move(self.id, self.x, self.y)
❷       pos = self.canvas.coords(self.id)
❸       if pos[1] <= 0:
            self.y = 1
❹       if pos[3] >= self.hauteur_canevas:
            self.y = -1
```

En ❶, nous modifions l'appel à la fonction `move` pour lui passer les variables d'objet `x` et `y` au lieu de valeurs numériques brutes. Ensuite, nous créons une variable `pos` (position) avec comme valeur le résultat de l'appel de la fonction `coords` du canevas (❷). Cette fonction renvoie les coordonnées *x* et *y* de tout objet dessiné à l'écran, du moment que nous en connaissons le numéro d'identifiant. Ici, nous passons à `coords` la variable d'objet `id`, qui contient l'identifiant de l'ovale.

La fonction `coords` renvoie les coordonnées sous la forme d'une liste de quatre nombres. Si nous affichons les résultats de l'appel de cette fonction, nous voyons quelque chose du type :

```
print(self.canvas.coords(self.id))
[255.0, 29.0, 270.0, 44.0]
```

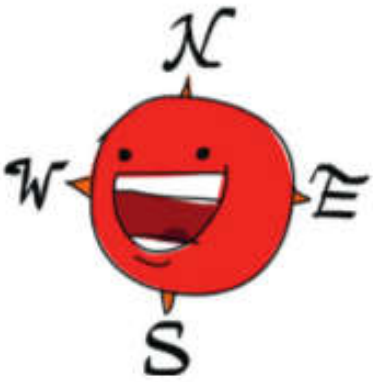
Les deux premiers nombres de la liste, `255.0` et `29.0`, correspondent aux coordonnées du « coin » supérieur de l'ovale (*x1* et *y1*), tandis que les deux derniers, `270.0` et `44.0`, sont celles du coin inférieur droit (*x2* et *y2*). Nous exploitons ces valeurs dans les lignes suivantes du programme.

En ❸, nous vérifions que la coordonnée *y1* (qui correspond au sommet de la balle) est inférieure ou égale à 0 (le bord haut de l'écran). Si c'est le cas, nous forçons la variable d'objet `y` à `1`. En pratique, cela signifie que, si la balle touche le haut de l'écran, nous cessons de soustraire 1 à la position verticale, donc nous arrêtons d'aller vers le haut.

De la même manière, nous vérifions que la coordonnée *y2*, c'est-à-dire le bas de la balle, est supérieure ou égale à la variable `hauteur_canevas` (❹). Si elle l'est, alors nous ramenons la variable d'objet `y` à `-1`.

Exécute le programme à nouveau pour constater que la balle rebondit en haut et en bas, jusqu'à ce que tu fermes la fenêtre.

Changer la direction de départ de la balle



Nous avons obtenu une balle qui rebondit doucement de haut en bas, mais cela ne suffit toujours pas pour faire un jeu. Nous allons donc améliorer un peu les choses et modifier la direction de départ de la balle, c'est-à-dire l'angle de déplacement de celle-ci au départ du jeu. Dans la fonction `__init__` de la classe `Balle`, modifie ces lignes :

```
self.x = 0
self.y = -1
```

Remplace-les par les suivantes (vérifie que tu as bien laissé les nombres d'espaces corrects, il doit y en avoir huit, au début de chaque ligne) :

```
❶        departs = [-3, -2, -1, 1, 2, 3]
❷        random.shuffle(departs)
❸        self.x = departs[0]
❹        self.y = -3
```

Nous créons d'abord (❶) la variable `departs` avec une liste de six nombres, que nous mélangeons par l'appel à `random.shuffle` (❷). Nous donnons à `x` la valeur du premier élément de la liste (❸), de sorte que celui-ci peut prendre n'importe quelle valeur au hasard dans cette liste, de `-3` à `3`.

Si nous modifions la valeur de `y` en `-3` pour accélérer la balle (❹), nous devons aussi apporter quelques ajouts pour nous assurer que la balle ne disparaît pas par les côtés de l'écran. Ajoute les lignes suivantes à la fin de la fonction `__init__` pour mémoriser la largeur du canevas dans une nouvelle variable d'objet, `largeur_canevas` :

```
        self.largeur_canevas = self.canvas.winfo_width()
```

Nous utilisons cette nouvelle variable d'objet dans la fonction `dessiner` pour vérifier si la balle touche le côté gauche ou droit du canevas :

```
        if pos[0] <= 0:
            self.x = 3
        if pos[2] >= self.largeur_canevas:
            self.x = -3
```

Comme nous réglons `x` à `3` et `-3`, nous faisons de même avec `y`, pour que la balle se déplace à la même vitesse dans toutes les directions. La fonction `dessiner` devient la suivante :

```
    def dessiner(self):
        self.canvas.move(self.id, self.x, self.y)
        pos = self.canvas.coords(self.id)
        if pos[1] <= 0:
            self.y = 3
        if pos[3] >= self.hauteur_canevas:
            self.y = -3
        if pos[0] <= 0:
            self.x = 3
        if pos[2] >= self.largeur_canevas:
            self.x = -3
```

Enregistre et exécute le programme. La balle doit désormais rebondir dans tout l'écran sans disparaître. Le programme complet, à ce stade, est celui-ci :

```
from tkinter import *
import random
import time

class Balle:
    def __init__(self, canvas, couleur):
        self.canvas = canvas
        self.id = canvas.create_oval(10, 10, 25, 25, fill=couleur)
        self.canvas.move(self.id, 245, 100)
        departs = [-3, -2, -1, 1, 2, 3]
        random.shuffle(departs)
        self.x = departs[0]
        self.y = -3
        self.hauteur_canevas = self.canvas.winfo_height()
        self.largeur_canevas = self.canvas.winfo_width()

    def dessiner(self):
        self.canvas.move(self.id, self.x, self.y)
        pos = self.canvas.coords(self.id)
        if pos[1] <= 0:
            self.y = 3
        if pos[3] >= self.hauteur_canevas:
            self.y = -3
        if pos[0] <= 0:
            self.x = 3
        if pos[2] >= self.largeur_canevas:
            self.x = -3

tk = Tk()
tk.title("Jeu")
tk.resizable(0, 0)
tk.wm_attributes("-topmost", 1)
canvas = Canvas(tk, width=500, height=400, bd=0, highlightthickness=0)
canvas.pack()
tk.update()


balle = Balle(canvas, 'red')

while 1:
    balle.dessiner()
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

Ce que tu as appris

Dans ce chapitre, nous avons débuté notre premier jeu à l'aide du module `tkinter`. Nous avons créé une classe pour la balle et l'avons animée pour qu'elle se déplace sur tout l'écran. Grâce aux coordonnées, nous avons vérifié que la balle touche bien les bords du canevas, puis nous avons fait en sorte qu'elle rebondisse. Nous avons également tiré parti de la fonction `shuffle` du module `random` pour que notre balle ne parte pas toujours exactement dans la même direction. Au chapitre suivant, nous ajoutons la raquette pour compléter ce jeu.

14

ACHEVER TON PREMIER JEU : REBONDIR !

Au chapitre précédent, nous avons débuté la création d'un tout premier jeu, Rebondir ! Nous avons créé le canevas et lui avons ajouté une balle magique, mais cette balle rebondit à l'infini à l'écran, tant que tu ne fermes pas la fenêtre. Ceci n'a pas vraiment d'intérêt. Alors, dans ce chapitre, nous allons ajouter une raquette pour qu'un utilisateur puisse renvoyer la balle. Nous allons aussi ajouter un élément de chance, pour épicer un peu le jeu et apporter plus d'amusement.

Ajouter la raquette

Comme il n'y a pas vraiment d'amusement à voir rebondir une balle à l'infini, nous allons impliquer l'utilisateur en lui offrant une raquette.

Commence par ajouter le code qui suit pour créer la classe `Raquette`. Entre ces instructions deux lignes après la fonction `dessiner` de la classe `Balle`.

```
    def dessiner(self):
        self.canvas.move(self.id, self.x, self.y)
        pos = self.canvas.coords(self.id)
        if pos[1] <= 0:
            self.y = 3
        if pos[3] >= self.hauteur_canevas:
            self.y = -3
        if pos[0] <= 0:
            self.x = 3
        if pos[2] >= self.largeur_canevas:
            self.x = -3

class Raquette:
    def __init__(self, canvas, couleur):
        self.canvas = canvas
        self.id = canvas.create_rectangle(0, 0, 100, 10, fill=couleur)
        self.canvas.move(self.id, 200, 300)

    def dessiner(self):
        pass
```

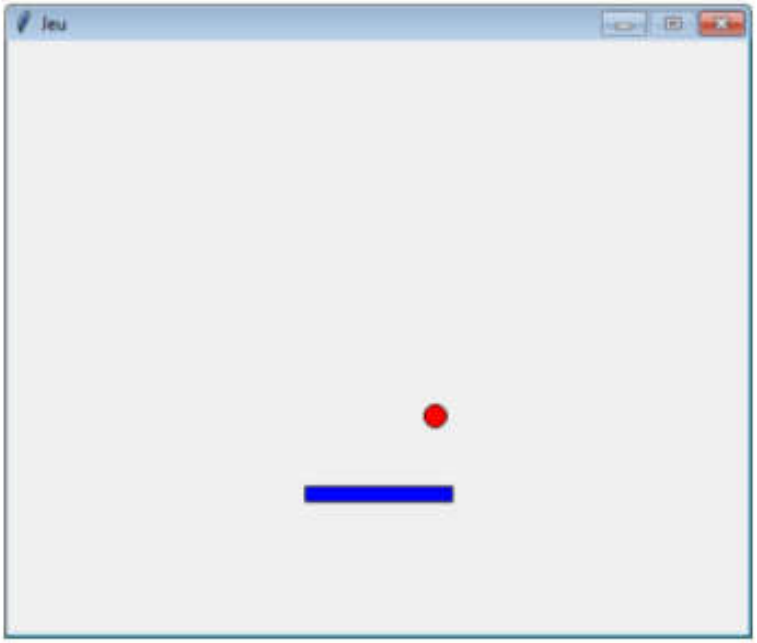
Le code ajouté ressemble fort à celui du début de la classe `Balle`, sauf qu'ici, nous appelons `create_rectangle` au lieu de `create_oval` et que nous plaçons le rectangle à 200 pixels du bord intérieur gauche du canevas et à 300 pixels du bord supérieur.

Quasiment à la fin du programme, juste avant la création de l'objet `balle`, crée ensuite un objet de la classe `Raquette` (de couleur bleue), puis ajoute dans la boucle principale un appel à la fonction `dessiner` de la `raquette`, comme suit :

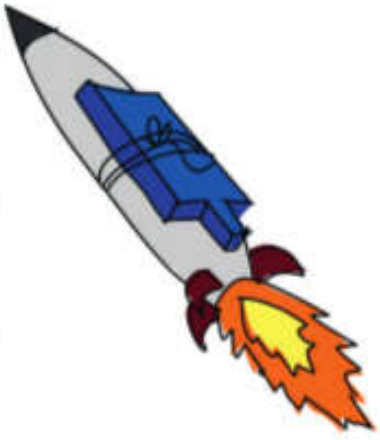
```
raquette = Raquette(canvas, 'blue')
balle = Balle(canvas, 'red')

while 1:
    balle.dessiner()
    raquette.dessiner()
```

```
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

À partir de là, lorsque tu exécutes le programme, tu vois une balle qui rebondit sur les côtés et un rectangle stationnaire (qui ne bouge pas).



Déplacer la raquette



Pour que la raquette se déplace, nous utilisons la liaison d'événement pour associer les touches de flèches `Gauche` et `Droite` à de nouvelles fonctions de la classe `Raquette`. Quand l'utilisateur appuie sur la flèche gauche, la variable `x` est réglée à `-2` (donc pour déplacer la raquette vers la gauche). Une pression sur la touche de la flèche droite règle la variable `x` à `2` (vers la droite).

La première étape consiste à ajouter la variable d'objet `x` à la fonction `__init__` de la classe `Raquette`, ainsi qu'une variable pour la largeur du canevas, comme celle que nous avions définie dans la classe `Balle` :

```
    def __init__(self, canvas, couleur):
        self.canvas = canvas
        self.id = canvas.create_rectangle(0, 0, 100, 10, fill=couleur)
        self.canvas.move(self.id, 200, 300)
```

```
        self.x = 0
        self.largeur_canevas = self.canvas.winfo_width()
```

Comme nous devons changer la direction vers la gauche (`vers_gauche`) et vers la droite (`vers_droite`) à l'aide de fonctions, nous les ajoutons juste avant la fonction `dessiner` :

```
    def vers_gauche(self, evt):
        self.x = -2

    def vers_droite(self, evt):
        self.x = 2
```

Nous pouvons ensuite associer ces fonctions aux touches adéquates dans la fonction `__init__` de la classe en deux lignes. Nous avons déjà utilisé la liaison d'un événement à une action (voir la section « Réagir à un événement » du chapitre 12, où Python appelle une fonction lors de la pression d'une touche du clavier). Dans ce cas-ci, nous lions la fonction `vers_gauche` de la classe `Raquette` à la pression sur la touche flèche `Gauche` à l'aide du nom d'événement `'<KeyPress-Left>'`. De même, nous associons la fonction `vers_droite` à la touche `Droite` à l'aide de l'événement `'<KeyPress-Right>'`. Dès lors, notre fonction `__init__` devient :

```
    def __init__(self, canvas, couleur):
        self.canvas = canvas
        self.id = canvas.create_rectangle(0, 0, 100, 10, fill=couleur)
        self.canvas.move(self.id, 200, 300)
        self.x = 0
        self.largeur_canevas = self.canvas.winfo_width()
        self.canvas.bind_all('<KeyPress-Left>', self.vers_gauche)
        self.canvas.bind_all('<KeyPress-Right>', self.vers_droite)
```

Quant à la fonction `dessiner` de la classe `Raquette`, elle s'inspire fortement de celle de la classe `Balle` :

```
    def dessiner(self):
❶       self.canvas.move(self.id, self.x, 0)
❷       pos = self.canvas.coords(self.id)
❸       if pos[0] <= 0:
            self.x = 0
❹       elif pos[2] >= self.largeur_canevas:
            self.x = 0
```

La fonction `move` ❶ du canevas permet de déplacer la raquette dans la direction donnée par la variable `x`. Le zéro en troisième para-

mètre signifie qu'il n'y a pas de déplacement vertical. Nous lisons ensuite les coordonnées d'emplacement de la raquette ❷ pour vérifier si elle a touché le côté gauche d'abord, droit ensuite, de l'écran, grâce à la valeur contenue dans `pos`.

Au lieu de rebondir comme la balle, la raquette doit simplement s'arrêter de bouger. Donc, quand la coordonnée *x* de gauche (`pos[0]`) est inférieure ou égale à 0 (`<= 0`), nous réglons la variable `x` à `0` ❸. De mêm, quand la coordonnée *x* de droite (`pos[2]`) est supérieure ou égale à la largeur du canevas, c'est-à-dire `>= self.largeur_canevas`, nous réglons aussi la variable `x` à `0` ❹.

NOTE *Si tu exécutes le programme maintenant, tu dois cliquer dans le canevas avant que le jeu ne reconnaisse les actions sur les touches* `Gauche` *et* `Droite`. *Quand tu cliques dans le canevas, tu donnes le focus au canevas, ce qui signifie qu'il sait alors qu'il doit prendre en charge les pressions de l'utilisateur sur les touches du clavier.*

Détecter quand la balle touche la raquette

À ce point de la programmation, la balle ne heurte pas la raquette, mais elle la traverse, tout simplement. Or, la balle doit savoir quand elle a touché la raquette, au même titre qu'elle sait qu'elle a touché un des murs.

Nous pourrions résoudre ce problème en ajoutant du code à la fonction `dessiner` (là où le code détecte les murs), mais une meilleure idée consiste à déplacer ce genre de code dans de nouvelles fonctions, pour découper les tâches dans de plus petits extraits. Si nous plaçons trop de code au même endroit (dans une seule fonction, par exemple), le programme devient plus difficile à comprendre. Apportons les modifications nécessaires.

D'abord, modifions la fonction `__init__` de la balle pour que nous puissions lui passer l'objet `raquette` en paramètre :

```
    classe Balle:
❶     def __init__(self, canvas, raquette, couleur):
            self.canvas = canvas
❷           self.raquette = raquette
            self.id = canvas.create_oval(10, 10, 25, 25, fill=couleur)
            self.canvas.move(self.id, 245, 100)
            departs = [-3, -2, -1, 1, 2, 3]
            random.shuffle(departs)
```

```
        self.x = departs[0]
        self.y = -3
        self.hauteur_canevas = self.canvas.winfo_height()
        self.largeur_canevas = self.canvas.winfo_width()
```

En ❶, nous modifions les paramètres de la fonction `__init__` pour y inclure la raquette. En ❷, nous affectons le paramètre `raquette` à la variable d'objet `raquette`.

L'objet `raquette` mémorisé, nous devons modifier le code où nous avons créé l'objet `balle`, puisqu'un nouveau paramètre y apparaît. La modification a lieu vers la fin du programme, avant la boucle principale :

```
raquette = Raquette(canvas, 'blue')
balle = Balle(canvas, raquette, 'red')

while 1:
    balle.dessiner()
    raquette.dessiner()
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

Le code nécessaire pour vérifier si la balle a heurté la raquette est un peu plus compliqué que celui qui détecte les murs. Nous nommons cette fonction `heurter_raquette` et l'ajoutons à la fonction `dessiner` de la classe `Balle`, là où nous vérifions si la balle touche le bas de l'écran :

```
    def dessiner(self):
        self.canvas.move(self.id, self.x, self.y)
        pos = self.canvas.coords(self.id)
        if pos[1] <= 0:
            self.y = 3
        if pos[3] >= self.hauteur_canevas:
            self.y = -3
        if self.heurter_raquette(pos) == True:
            self.y = -3
        if pos[0] <= 0:
            self.x = 3
        if pos[2] >= self.largeur_canevas:
            self.x = -3
```

Ainsi, si `heurter_raquette` retourne vrai, alors nous changeons la direction verticale, c'est-à-dire la variable d'objet `y` à `-3`. N'essaie pas

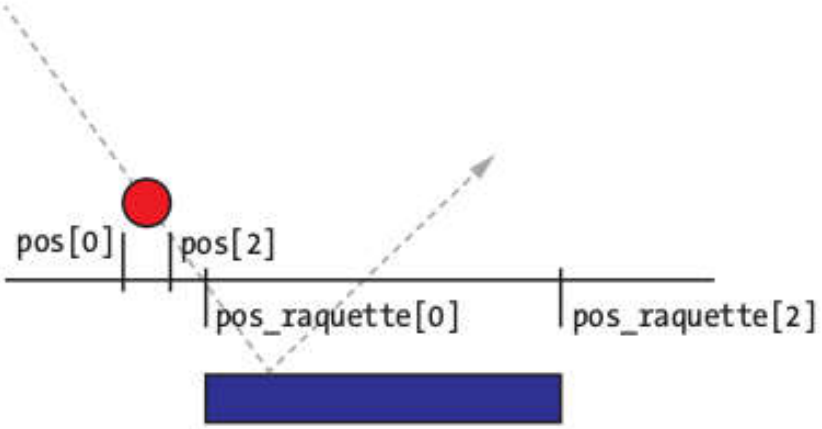
d'exécuter le programme à ce stade car la fonction `heurter_raquette` n'existe pas encore. Créons-la maintenant.

Ajoute la fonction `heurter_raquette` juste avant la fonction `dessiner`.

```
❶    def heurter_raquette(self, pos):
❷        pos_raquette = self.canvas.coords(self.raquette.id)
❸        if pos[2] >= pos_raquette[0] and pos[0] <= pos_raquette[2]:
❹            if pos[3] >= pos_raquette[1] and pos[3] <= pos_raquette[3]:
                 return True
         return False
```

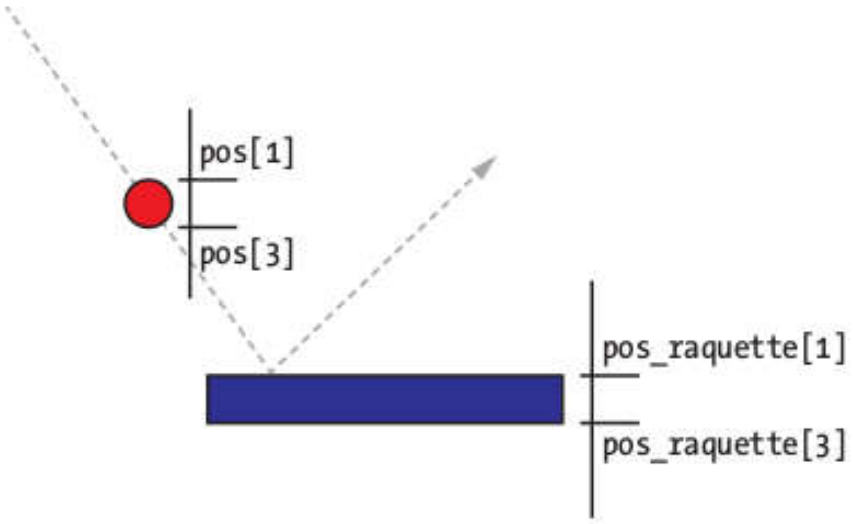
Nous définissons d'abord la fonction avec le paramètre `pos` (position) ❶. Cette ligne contient les coordonnées actuelles de la balle. Puis, nous prenons les coordonnées de la raquette et nous les mémorisons dans la variable `pos_raquette` ❷.

En ❸, nous avons la première partie de notre instruction `if` et nous disons « si le côté droit de la balle est plus grand que le côté gauche de la raquette et si le côté gauche de la balle est plus petit que le côté droit de la raquette, alors… ». Dans cette ligne, `pos[2]` contient la coordonnée *x* du côté droit de la balle et `pos[0]` contient celle de son côté gauche. En revanche, `pos_raquette[0]` contient la coordonnée *x* du côté gauche de la raquette et `pos_raquette[2]` contient celle de son côté droit. Le diagramme suivant montre mieux comment ces coordonnées se présentent quand la balle est sur le point de toucher la raquette.



La balle tombe sur la raquette et, à cet instant précis, tu vois clairement que le côté droit de la balle (`pos[2]`) n'a pas encore dépassé le côté gauche de la raquette (donc `pos_raquette[0]`).

Nous vérifions ensuite ❹ si le bas de la balle (`pos[3]`) se trouve entre le haut de la raquette (`pos_raquette[1]`) et son bas (`pos_raquette[3]`). Le diagramme suivant montre que le bas de la balle (`pos[3]`) doit encore atteindre le haut de la raquette (`pos_raquette[1]`).



Ainsi, selon cet emplacement actuel de la balle, la fonction `heurter_raquette` renvoie faux.

NOTE *Pourquoi faut-il vérifier si le dessous de la balle est entre le haut et le bas de la raquette ? Pourquoi ne pas simplement regarder si le bas de la balle a heurté le haut de la raquette ? Ceci est dû au fait que, chaque fois que nous déplaçons la balle dans le canevas, nous nous déplaçons par sauts de trois pixels à la fois. Si nous ne vérifions que le fait que la balle a atteint le haut de la raquette (`pos[1]`), il se peut que nous ayons déjà sauté au-delà de cet emplacement. Dans ce cas, la balle poursuivrait son voyage et elle traverserait la raquette sans s'arrêter.*

Ajouter un facteur chance

Pour que ce programme devienne un véritable jeu et ne se limite pas à une raquette et une balle rebondissante, il doit apporter un facteur chance, c'est-à-dire une possibilité pour l'utilisateur de perdre. Dans le jeu tel qu'il est actuellement, la balle rebondit à l'infini donc il n'y a rien à perdre.

Pour achever notre jeu, nous allons ajouter du code pour que la partie se termine quand la balle touche le bas du canevas, autrement dit quand la balle touche le sol.

En premier lieu, nous ajoutons la variable d'objet `touche_bas` à la fin de la fonction `__init__` de la classe `Balle` :

```
        self.hauteur_canevas = self.canvas.winfo_height()
        self.largeur_canevas = self.canvas.winfo_width()
        self.touche_bas = False
```

Ensuite, nous devons modifier un peu la boucle principale, au bas du programme, comme ceci :

```
while 1:
    if balle.touche_bas == False:
        balle.dessiner()
        raquette.dessiner()
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

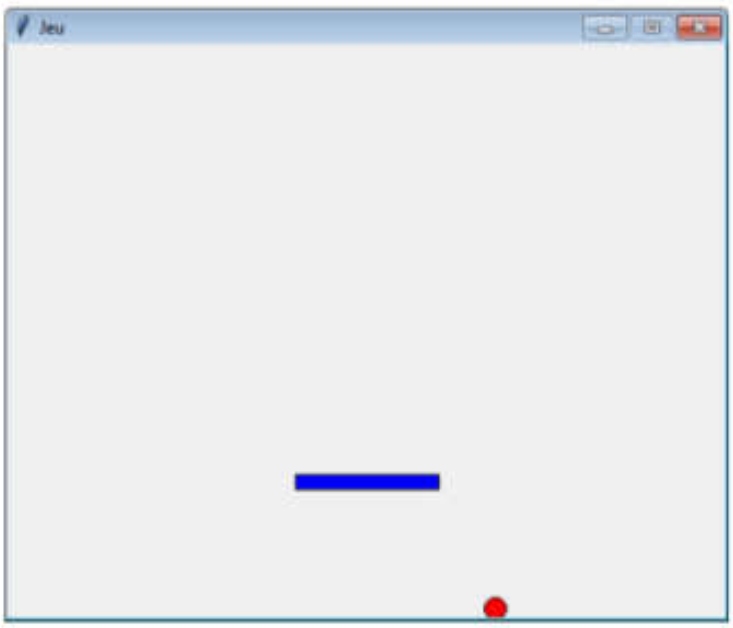
À partir de maintenant, la boucle vérifie à chaque passage la valeur de `touche_bas`, pour savoir si la balle a touché le bas de l'écran. Le code continue de déplacer la balle et la raquette seulement si la balle n'a pas touché le bas, comme l'indique l'instruction `if`. La partie se termine quand la balle et la raquette cessent de se déplacer car nous ne les animons plus.

La modification finale concerne la fonction `dessiner` de la classe `Balle` :

```
    def dessiner(self):
        self.canvas.move(self.id, self.x, self.y)
        pos = self.canvas.coords(self.id)
        if pos[1] <= 0:
            self.y = 3
        if pos[3] >= self.hauteur_canevas:
            self.touche_bas = True
        if self.heurter_raquette(pos) == True:
            self.y = -3
        if pos[0] <= 0:
            self.x = 3
        if pos[2] >= self.largeur_canevas:
            self.x = -3
```

Nous avons corrigé l'instruction `if` qui teste si la balle a touché le bas de l'écran, c'est-à-dire si sa position en *y2* est supérieure ou égale à la `hauteur_canevas`. Si c'est le cas, à la ligne suivante, nous réglons `touche_bas` à vrai, au lieu de changer la valeur de la variable `y`, parce qu'il n'est plus nécessaire de la faire rebondir dès qu'elle a touché le bas de l'écran.

À présent, quand tu exécutes le programme de jeu et rate la balle avec la raquette, tous les mouvements de l'écran s'arrêtent et la partie se termine quand la balle touche le bas du canevas :



Le programme complet a désormais l'allure suivante. S'il ne fonctionne pas comme prévu, compare ton code à celui-ci pour trouver les éventuelles erreurs.

```
from tkinter import *
import random
import time

class Balle:
    def __init__(self, canvas, raquette, couleur):
        self.canvas = canvas
        self.raquette = raquette
        self.id = canvas.create_oval(10, 10, 25, 25, fill=couleur)
        self.canvas.move(self.id, 245, 100)
        departs = [-3, -2, -1, 1, 2, 3]
        random.shuffle(departs)
        self.x = departs[0]
        self.y = -3
        self.hauteur_canevas = self.canvas.winfo_height()
        self.largeur_canevas = self.canvas.winfo_width()
        self.touche_bas = False

    def heurter_raquette(self, pos):
        pos_raquette = self.canvas.coords(self.raquette.id)
        if pos[2] >= pos_raquette[0] and pos[0] <= pos_raquette[2]:
            if pos[3] >= pos_raquette[1] and pos[3] <= pos_raquette[3]:
                return True
        return False
```

```
    def dessiner(self):
        self.canvas.move(self.id, self.x, self.y)
        pos = self.canvas.coords(self.id)
        if pos[1] <= 0:
            self.y = 3
        if pos[3] >= self.hauteur_canevas:
            self.touche_bas = True
        if self.heurter_raquette(pos) == True:
            self.y = -3
        if pos[0] <= 0:
            self.x = 3
        if pos[2] >= self.largeur_canevas:
            self.x = -3

class Raquette:
    def __init__(self, canvas, couleur):
        self.canvas = canvas
        self.id = canvas.create_rectangle(0, 0, 100, 10, fill=couleur)
        self.canvas.move(self.id, 200, 300)
        self.x = 0
        self.largeur_canevas = self.canvas.winfo_width()
        self.canvas.bind_all('<KeyPress-Left>', self.vers_gauche)
        self.canvas.bind_all('<KeyPress-Right>', self.vers_droite)

    def vers_gauche(self, evt):
        self.x = -2

    def vers_droite(self, evt):
        self.x = 2

    def dessiner(self):
        self.canvas.move(self.id, self.x, 0)
        pos = self.canvas.coords(self.id)
        if pos[0] <= 0:
            self.x = 0
        elif pos[2] >= self.largeur_canevas:
            self.x = 0

tk = Tk()
tk.title("Jeu")
tk.resizable(0, 0)
tk.wm_attributes("-topmost", 1)
canvas = Canvas(tk, width=500, height=400, bd=0, highlightthickness=0)
canvas.pack()
tk.update()

raquette = Raquette(canvas, 'blue')
balle = Balle(canvas, raquette, 'red')
```

```
while 1:
    if balle.touche_bas == False:
        balle.dessiner()
        raquette.dessiner()
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

Ce que tu as appris

Avec ce chapitre, nous avons terminé la réalisation de notre premier jeu à l'aide du module `tkinter`. Nous avons créé la classe de la raquette et exploité les coordonnées pour vérifier si la balle heurte la raquette ou les murs du canevas du jeu. Grâce à la liaison d'événement à des actions, nous avons associé les touches de flèches `Gauche` et `Droite` pour contrôler les mouvements de la raquette. La boucle principale appelle les fonctions `dessiner` des deux objets pour les animer. Enfin, nous avons modifié le code pour ajouter un petit facteur de chance au jeu : quand le joueur rate la balle, la partie se termine au moment où elle touche le bord inférieur du canevas.

Puzzles de programmation

Pour l'instant, notre jeu demeure un peu simple. Il n'y a pas grand-chose à changer pour créer un programme professionnel de jeu. Essaie d'améliorer le code selon les suggestions suivantes pour le rendre plus intéressant. Compare ensuite tes réalisations à celles du site d'accompagnement du livre.

1. Retarder le début du jeu

Le jeu démarre un peu rapidement et il faut cliquer sur le canevas pour qu'il commence à reconnaître les touches du clavier. Peux-tu ajouter un délai au démarrage de la partie, pour donner au joueur suffisamment de temps pour cliquer sur le canevas ? Mieux, peux-tu ajouter une liaison d'événement à un clic de la souris pour ne lancer la partie qu'au moment du clic ?

Conseil 1 : tu as déjà ajouté des liaisons d'événements à la classe `raquette`, donc c'est un bon endroit pour commencer tes recherches.

Conseil 2 : l'événement à détecter pour le bouton gauche de la souris est la chaîne `'<Button-1>'`.

2. Un véritable « Partie terminée »

Pour l'instant, le canevas se fige quand la partie se termine, ce qui n'est pas très convivial. Essaie d'ajouter le texte « Partie terminée » quand la balle touche le bord inférieur de l'écran. Tu peux utiliser la fonction `create_text`, mais tu peux trouver aussi une piste du côté du paramètre nommé `state` (état), qui prend des valeurs comme `normal` et `hidden` (caché). Cherche du côté d'`itemconfig` (voir la section « Autres façons d'utiliser l'identifiant » du chapitre 12). En guise de défi supplémentaire, ajoute un délai pour que le texte n'apparaisse pas immédiatement.

3. Accélérer la balle

Si tu as déjà joué au tennis, tu sais que quand la balle touche ta raquette, elle part parfois plus vite qu'elle n'est arrivée, selon la force de ton mouvement. La balle de notre jeu évolue actuellement à sa propre vitesse, que la raquette bouge ou pas. Essaie d'améliorer le programme pour que la vitesse de déplacement de la raquette soit communiquée à celle de la balle.

4. Enregistrer le score du joueur

Et si on parlait des scores ? Il est facile de compter au fur et à mesure le nombre de fois que la balle touche la raquette. Essaie d'afficher le score dans le coin supérieur droit du canevas. Cherche également du côté d'`itemconfig` (toujours au chapitre 12) pour une piste vers la solution.