DojoBoy – Pong

Cet atelier a pour objectif de t'examiner les bases de la programmation en Micropython (Python) de la console DojoBoy et de créer un jeu inspiré du jeu Pong.

Au terme de celui-ci, tu seras capable de :

- Initialiser la console
- Afficher un objet à l'écran
- Déplacer un objet à l'écran
- Faire une boucle finie et infinie
- Utiliser les boutons de la console
- Définir une fonction

Pour comprendre les étapes de cet atelier, il est préférable que tu aies déjà appris les bases de la programmation avec Scratch et que tu comprennes ce qu'est une « variable », une « condition » et une « boucle ».

Le code dans ces pages n'est pas complet. Il est indispensable d'ajouter et adapter le code pour arriver à l'objectif. Fais-toi aider des coachs si tu es perdu ou que tu ne comprends pas !

Les challenge en fin d'étapes vont te permette d'expérimenter ton code et de mieux le comprendre. Ne les passe pas trop vite... 😉

Initialisation du DojoBoy

L'accès aux fonctions qui permettent d'utiliser l'écran, les boutons et le son de la console sont rassemblés dans un « objet ».

Tous les programmes que tu vas écrire pour la console devront commencer par « importer » cet objet (dire au programme de rendre disponible l'objet). Il faut donc commencer les programmes par les lignes suivantes.

```
from dojoboy_v1 import DojoBoy
```

Cette ligne signifie qu'il faut importer l'objet « Dojo Boy » qui se trouve dans le module « dojoboy_v1 ». Nous examinerons dans un autre atelier ce qu'est un « module » et un « objet » dans le langage Python. Ne t'y attarde pas maintenant 😉

Il faut ensuite « instancier » cet objet (créer cet objet et lui donner un nom) pour ensuite l'utiliser dans nos programmes. Lors de la création de cet objet, des paramètres de configuration seront ajoutés pour configurer l'écran et les boutons de la console.

Quelques paramètres sont disponibles pour définir des paramètres particuliers.

Il n'est pas utile de s'étendre sur tous ces paramètres. Un seul est pratique pour débuter, il s'agit du paramètre pour activer ou désactiver l'écran d'accueil au démarrage d'un programme.

Pour créer l'objet DojoBoy **avec** l'écran d'accueil, il faut utiliser la commande suivante :

```
djb = DojoBoy()
```

Pour l'objet DojoBoy **sans** l'écran d'accueil:

```
djb=DojoBoy(show_intro=False)
```

Maintenant, commençons à programmer la console ! 👍

Etape 1 : Premier programme...

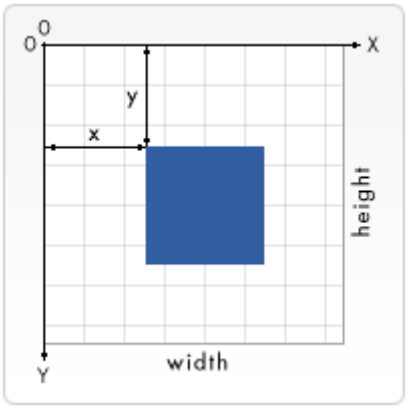
Commençons par examiner comment afficher une « balle » à l'écran...

Les premières commandes du programme vont définir la position où la balle doit être affichée sur l'écran.

Nous allons donc définir les variables **pos_x** et **pos_y** et les initialiser avec les valeurs de la position sur la grille de pixels de l'écran.

L'illustration suivante représente l'écran de la console. Le pixel à la position 0,0 se trouve en haut à gauche. L'axe des X est horizontal (la largeur de l'écran, « width » en anglais) et l'axe Y est vertical (la hauteur de l'écran, « height » en anglais).

L'écran du DojoBoy est composé suivant le type d'écran de 240 ou 160 points sur la largeur et 240 ou 128 points sur la hauteur.



Pour afficher la balle au milieu de l'écran 160x128 points, il faut donc initialiser les variables de la manière suivante :

```
pos_x = 80
pos_y = 64
```

Il faut ensuite définir le diamètre en point (pixel) de la balle.

```
ball_size = 3 # ball radius
```

On peut remarquer en fin de ligne le caractère **#** qui permet d'ajouter des commentaires dans le programme. Tout ce qui suit ce caractère sur une même ligne n'est pas pris en compte dans le programme.

Les lignes suivantes vont ensuite afficher une balle au centre de l'écran.

```
djb.display.ellipse(pos_x, pos_y, ball_size, ball_size, djb.display.RED_H, True) #draw ball
djb.display.show() # show screen
```

La fonction « **djb.display.ellipse** » signifie « dessine sur l'écran de la console » (djb.display) « un ovale » (ellipse). Ici, c'est un ovale particulier car il a la forme d'un cercle.

Les paramètres qui suivent se basent sur les variables définies avant.

- **pos_x** : position du centre du cercle sur l'axe X
- **pos_y** : position du center du cercle sur l'axe Y
- **ball_size,ball_size** : défini le rayon du cercle
- **djb.display.RED_H** : défini la couleur de cercle
- **True** : indique le cercle doit être rempli de couleur

La fonction précédente ne montre pas directement le contenu de l'écran. Il faut utiliser la fonction **djb.display.show()** pour cela.

Challenge

- Change la couleur de la balle
- Change la taille de la balle
- Affiche la balle dans le haut de l'écran
- Affiche la balle à gauche de l'écran
- Affiche trois balles réparties sur l'écran

Etape 2 : Déplacement d'une forme sur l'écran

Le début du programme est identique à celui de l'étape précédente.

```
pos_x = 80
pos_y = 64
ball_size = 3 # ball radius

djb.display.fill(djb.display.BLACK)
```

Cette ligne permet de remplir l'écran (**fill** en anglais) avec de la couleur noire.

```
for x in range (0,150,1): # loop from 0 to 150 by step 1
    djb.display.ellipse(x, pos_y, ball_size, ball_size, djb.display.RED_H, True) #draw ball
    djb.display.show() # show screen
```

Ces lignes permettent de créer une boucle (**for** en anglais). La variable x est donc incrémentée (augmentée) d'**une** unité en commençant à **0** jusque **150** (**range (0,150,1**).

Il faut remarquer que les lignes suivantes sont décalées vers la droite. Ce décalage permet de définir les commandes qui font partie de la boucle. Ce décalage s'appelle une « indentation ». La façon la plus simple pour créer cette indentation est d'utiliser la touche **Tabulation** (à gauche de la touche A sur le clavier).

Challenge

- Adapte le code pour afficher la balle sans sa trace
- Augmente la vitesse de la balle
- Déplace la balle en diagonale
- Déplace la balle en diagonale mais plus rapidement horizontalement que verticalement

Etape 3 : Déplacement sans fin de la balle

Adapte le programme de l'étape précédente avec les lignes suivantes.

```
pos_x = 80
pos_y = 64
ball_size = 3 # ball radius

while True:
    pos_x += 1
    djb.display.ellipse(pos_x, pos_y, ball_size, ball_size, djb.display.RED_H, True) #draw ball
    djb.display.show() # show screen
```

La différence avec le programme précédent est la ligne avec la commande **while** (« Tant que » en français). Associé avec la condition **True**, cette commande permet de créer une boucle infinie (qui ne s'arrête jamais). Comme pour la boucle **for**, les lignes suivantes font partie de la boucle et sont donc indentés.

Challenge

- Adapte le code pour afficher la balle de gauche à droite et de droite à gauche en la faisant rebondir sur le côté droit et gauche
- Dessine un rectangle bleu sur le périmètre de l'écran avec la commande **djb.display.rect(pos_x,pos_y,largeur,hauteur, couleur, remplissage)**.
- Adapte le code pour déplacer la balle à l'intérieur du rectangle en la faisant rebondir sur ses bords

Etape 4 : Utilisation des boutons

Déplaçons la balle avec les boutons directionnels

```
pos_x = 80
pos_y = 64
ball_size = 3 # ball radius

while True:
    djb.scan_jst_btn() #scan le joystick et les buttons
```

La ligne précédente permet de vérifier si le joystick a été déplacé ou si un bouton a été pressé. La ligne suivante permet de vérifier si le bouton indiqué a été pressé. Elle utilise la commande **if** (« si » en français). Ici, si le bouton directionnel droit est pressé ou si le joystick est déplacé vers la droite la commande de la ligne suivante est exécutée.

Remarque que cette ligne est indentée. Elle permet d'incrémenter (augmenter) d'une unité la variable **pos_x**. Cette ligne équivalente à **pos_x = pos_x + 1**.

```
    if djb.pressed(djb.btn_Right): # if right button pressed
        pos_x += 1
    djb.display.ellipse(pos_x, pos_y, ball_size, ball_size, djb.display.RED_H, True) #draw ball
    djb.display.show() # show screen
```

Challenge

- Adapte le code pour déplacer la balle à gauche et à droite avec les boutons directionnels ou le joystick.
- Adapte le code pour déplacer la balle dans toutes les directions avec les boutons directionnels

Etape 5 : Création d'une fonction

Améliorons le code avec l'utilisation des fonctions (méthodes).

Les deux lignes suivantes permettent de définir une fonction. La commande **def** est suivie du nom de la fonction (« **draw_ball(x,y)** »). L'utilisation d'une fonction permet d'éviter l'écriture répétitive d'un ensemble de commandes dans un programme. Cela permet aussi de rendre plus lisible ton programme.

Ici, **draw_ball(x,y)** permet de remplacer la commande pour dessiner le cercle à l'écran. Les paramètres **x** et **y** permettent de définir la position du cercle sur l'écran.

```
def draw_ball(x,y):
    djb.display.ellipse(x, y, ball_size, ball_size, djb.display.RED_H, True) #draw ball

while True:
    djb.scan_jst_btn() #scan buttons

    if djb.pressed(djb.btn_Right): # if right button pressed
        pos_x += 1

    draw_ball(pos_x,pos_y)

    djb.display.show() # show screen
```

Challenge

- Adapte le code pour déplacer la balle dans toutes les directions avec les boutons directionnels
- Adapte la fonction pour modifier la couleur de la balle lorsqu'on appuie sur le bouton A

Etape 6 : Changement de la vitesse de déplacement

Ajoutons une fonction pour gérer la vitesse de déplacement de la balle.

A côté des valeurs pour définir la position de la balle à l'écran (**pos_x** et **pos_y**), d'autres valeurs sont utilisées pour déterminer la vitesse (**velocity** en anglais) de déplacement sur l'axe X et Y(variables **velocity_x** et **velocity_y**). Plus ces valeurs sont élevées, plus le déplacement sera rapide.

La direction du déplacement de la balle sur les axes X et Y est déterminée par le signe **+** ou **–** appliqué aux variables de vitesse (exemple : **-velocity_x** ou **+velocity_y**).

Pour convertir la direction appliquée sur le joystick ou les boutons de direction, les variables de vitesse seront multipliées par **1** ou **-1**. Ainsi, pour un déplacement vers la droite, on multipliera la vitesse de l'axe X par 1. Vers le haut, on multipliera la vitesse de l'axe Y par -1.

```
pos_x = 80
pos_y = 64
ball_size = 3 # ball radius
velocity_x = 2
direction_x = 0

def draw_ball(x,y):
    djb.display.fill(djb.display.BLACK) # screen cleaning
    djb.display.ellipse(x, y, ball_size, ball_size, djb.display.RED_H, True) #draw ball

def move(p_x,p_y,vel_x,vel_y,dir_x,dir_y):
    p_x += vel_x * dir_x
    p_y += vel_y * dir_y
    return p_x,p_y

while True:
    direction_x = 0
    djb.scan_jst_btn() #scan buttons

    if djb.pressed(djb.btn_Right): # if right button pressed
        direction_x = 1

    pos_x,pos_y = move(pos_x,pos_y,velocity_x,0,direction_x,0)
    draw_ball(pos_x,pos_y)

    djb.display.show() # show screen
```

Challenge

- Adapte le code pour que la balle se déplace dans toutes les directions avec des rebonds sur le périmètre de l'écran
- Adapte le code pour diminuer la vitesse de la balle sur l'axe X avec le bouton gauche
- Adapte le code pour augmenter et diminuer la vitesse de la balle sur l'axe Y avec les boutons haut et bas

Etape 7 : Rebond sur les bords

Ajoutons une fonction pour gérer les rebonds de la balle sur un cadre

```
pos_x = 80
pos_y = 64
ball_size = 3 # ball radius
velocity_x = 1
velocity_y = 1
direction_x = -1
direction_y = 1
outline_x = 30 # pos x du coin superieur gauche du cadre
outline_y = 20 # pos y du coin superieur gauche du cadre
outline_w = 80 # largeur du cadre
outline_h = 60 # hauteur du cadre

def draw_ball(x,y):
        djb.display.fill(djb.display.BLACK) # screen cleaning
        # draw ball
        djb.display.ellipse(x, y, ball_size, ball_size, djb.display.RED_H, True)
        # draw border
        djb.display.rect(outline_x, outline_y, outline_w, outline_h, djb.display.BLUE_H)

def move(p_x,p_y,vel_x,vel_y,dir_x,dir_y):
        p_x += vel_x * dir_x
        p_y += vel_y * dir_y
        return p_x,p_y

def bounce(pos,dir,border_low,border_high): # rebond de la balle
        if pos <= border_low and dir == -1: # rebond gauche ou haut
        dir = 1
        elif pos >= border_high and dir == 1: # rebond droit ou bas
        dir = -1
        return dir

while True:

        djb.scan_jst_btn() #scan buttons

        if djb.pressed(djb.btn_Right): # if right button pressed
        direction_x = 1

        direction_x = bounce(pos_x,direction_x,outline_x,outline_x + outline_w)
        direction_y = bounce(pos_y,direction_y,outline_y,outline_y + outline_h)

        pos_x,pos_y = move(pos_x,pos_y,velocity_x,velocity_y,direction_x,direction_y)
        draw_ball(pos_x,pos_y)

        djb.display.show() # show screen
```

Etape 8 : Utilisation des contrôles de direction pour modifier la vitesse

```
pos_x = 80
pos_y = 64
ball_size = 3 # ball radius
velocity_x = 1
velocity_y = 1
direction_x = -1
direction_y = 1
outline_x = 30
outline_y = 20
outline_w = 80
outline_h = 60

def draw_ball(x,y):
        djb.display.fill(djb.display.BLACK) # screen cleaning
        # draw ball
        djb.display.ellipse(x, y, ball_size, ball_size, djb.display.RED_H, True)
        # draw border
        djb.display.rect(outline_x, outline_y, outline_w, outline_h, djb.display.BLUE_H)

def move(p_x,p_y,vel_x,vel_y,dir_x,dir_y):
        p_x += vel_x * dir_x
        p_y += vel_y * dir_y
        return p_x,p_y

def bounce(pos,dir,border_low,border_high): # rebond de la balle
        if pos <= border_low and dir == -1: # rebond gauche ou haut
        dir = 1
        elif pos >= border_high and dir == 1: # rebond droit ou bas
        dir = -1
        return dir

while True:
        djb.scan_jst_btn() #scan buttons

        if djb.pressed(djb.btn_Right) and velocity_x < 10: # if right button pressed
        velocity_x += 1
        if djb.pressed(djb.btn_Left) and velocity_x > 0: # if left button pressed
        velocity_x -= 1
        if djb.pressed(djb.btn_Up) and velocity_y < 10: # if up button pressed
        velocity_y += 1
        if djb.pressed(djb.btn_Down) and velocity_y > 0: # if down button pressed
        velocity_y -= 1

        direction_x = bounce(pos_x,direction_x,outline_x + ball_size + velocity_x,outline_x + ou-
tline_w - ball_size - velocity_x)
        direction_y = bounce(pos_y,direction_y,outline_y + ball_size + velocity_y,outline_y + ou-
tline_h - ball_size - velocity_y)

        pos_x,pos_y = move(pos_x,pos_y,velocity_x,velocity_y,direction_x,direction_y)
        draw_ball(pos_x,pos_y)

        djb.display.show_and_wait() # show screen and wait...
```

Etape 9 : On dessine la raquette...

```
pos_paddle_x = 80
pos_paddle_y = 118
paddle_w = 16
paddle_h = 8

pos_x = 80
pos_y = 64
ball_size = 3 # ball radius
velocity_x = 1
velocity_y = 1
direction_x = 1
direction_y = 1

outline_x = 30
outline_y = 20
outline_w = 100
outline_h = 60

def draw_border():  # draw border
        djb.display.rect(outline_x, outline_y, outline_w, outline_h, djb.display.BLUE_H)

def draw_ball(x,y):  # draw ball
        djb.display.ellipse(x, y, ball_size, ball_size, djb.display.RED_H, True)

def draw_paddle(x): # dessine la raquette
        djb.display.rect(x, pos_paddle_y, paddle_w, paddle_h, djb.display.GREEN_H, True)

def move(p_x,p_y,vel_x,vel_y,dir_x,dir_y):
        p_x += vel_x * dir_x
        p_y += vel_y * dir_y
        return p_x,p_y

def bounce(pos,dir,border_low,border_high): # rebond de la balle
        if pos <= border_low and dir == -1: # rebond gauche ou haut
        dir = 1
        elif pos >= border_high and dir == 1: # rebond droit ou bas
        dir = -1
        return dir

while True:
        djb.scan_jst_btn() #scan buttons

        if djb.pressed(djb.btn_Right) and velocity_x < 10: # if right button pressed
        velocity_x += 1
        if djb.pressed(djb.btn_Left) and velocity_x > 0: # if left button pressed
        velocity_x -= 1
        if djb.pressed(djb.btn_Up) and velocity_y < 10: # if up button pressed
        velocity_y += 1
        if djb.pressed(djb.btn_Down) and velocity_y > 0: # if down button pressed
        velocity_y -= 1

        direction_x = bounce(pos_x,direction_x,outline_x + ball_size + velocity_x,outline_x + ou-
tline_w - ball_size - velocity_x)
        direction_y = bounce(pos_y,direction_y,outline_y + ball_size + velocity_y,outline_y + ou-
tline_h - ball_size - velocity_y)
```

```
pos_x,pos_y = move(pos_x,pos_y,velocity_x,velocity_y,direction_x,direction_y)

draw_border()
draw_paddle(pos_paddle_x)
draw_ball(pos_x,pos_y)

djb.display.show_and_wait(True) # show screen and wait...
```

Etape 10

```
pos_paddle_x = 80
pos_paddle_y = 118
paddle_w = 16
paddle_h = 8
velocity_paddle = 2

pos_x = 80
pos_y = 64
ball_size = 3 # ball radius
velocity_x = 0
velocity_y = 0
direction_x = 1
direction_y = 1

outline_x = 0
outline_y = 20
outline_w = 120
outline_h = 108

def draw_border():
        djb.display.rect(outline_x, outline_y, outline_w, outline_h, djb.display.BLUE_H) # draw
border

def draw_ball(x,y):
        djb.display.ellipse(x, y, ball_size, ball_size, djb.display.RED_H, True) # draw ball

def draw_paddle(x):
        djb.display.rect(x, pos_paddle_y, paddle_w, paddle_h, djb.display.GREEN_H, True)

def move(p_x,p_y,vel_x,vel_y,dir_x,dir_y):
        p_x += vel_x * dir_x
        p_y += vel_y * dir_y
        return p_x,p_y

def bounce(pos,dir,border_low,border_high): # rebond de la balle
        if pos <= border_low and dir == -1: # rebond gauche ou haut
        dir = 1
        elif pos >= border_high and dir == 1: # rebond droit ou bas
        dir = -1
        return dir

while True:
        djb.scan_jst_btn() #scan buttons

        if djb.pressed(djb.btn_Right) and velocity_x < 10: # if right button pressed
        velocity_x += 1
        if djb.pressed(djb.btn_Left) and velocity_x > 0: # if left button pressed
        velocity_x -= 1
        if djb.pressed(djb.btn_Up) and velocity_y < 10: # if up button pressed
        velocity_y += 1
        if djb.pressed(djb.btn_Down) and velocity_y > 0: # if down button pressed
        velocity_y -= 1

        direction_x = bounce(pos_x,direction_x,outline_x + ball_size + velocity_x,outline_x + ou-
tline_w - ball_size - velocity_x)
```

```
        direction_y = bounce(pos_y,direction_y,outline_y + ball_size + velocity_y,outline_y + ou-
tline_h - ball_size - velocity_y)

        pos_x,pos_y = move(pos_x,pos_y,velocity_x,velocity_y,direction_x,direction_y)

        draw_border()
        draw_paddle(pos_paddle_x)
        draw_ball(pos_x,pos_y)

        djb.display.show_and_wait(True) # show screen and wait...
```

Etape 11 : Déplacement de la raquette

```
pos_paddle_x = 80
pos_paddle_y = 118
paddle_w = 16
paddle_h = 8
velocity_paddle = 2

pos_x = 80
pos_y = 64
ball_size = 3 # ball radius
velocity_x = 0
velocity_y = 0
direction_x = 1
direction_y = 1

outline_x = 0
outline_y = 20
outline_w = 120
outline_h = 108


def draw_border():
	djb.display.rect(outline_x, outline_y, outline_w, outline_h, djb.display.BLUE_H)

def draw_ball(x,y):
	djb.display.ellipse(x, y, ball_size, ball_size, djb.display.RED_H, True)

def draw_paddle(x):
	djb.display.rect(x, pos_paddle_y, paddle_w, paddle_h, djb.display.GREEN_H, True)

def move(p_x,p_y,vel_x,vel_y,dir_x,dir_y):
	p_x += vel_x * dir_x
	p_y += vel_y * dir_y
	return p_x,p_y

def bounce(pos,dir,border_low,border_high): # rebond de la balle
	if pos <= border_low and dir == -1: # rebond gauche ou haut
	dir = 1
	elif pos >= border_high and dir == 1: # rebond droit ou bas
	dir = -1
	return dir

while True:
	djb.scan_jst_btn() #scan buttons
	# if right button pressed
	if djb.pressed(djb.btn_Right) and pos_paddle_x < outline_x + outline_w - paddle_w:
	pos_paddle_x += velocity_paddle
	# if left button pressed
	if djb.pressed(djb.btn_Left) and pos_paddle_x > 0:
	pos_paddle_x -= velocity_paddle
	# if up button pressed
	if djb.pressed(djb.btn_Up) and velocity_y < 10:
	velocity_y += 1
	# if down button pressed
	if djb.pressed(djb.btn_Down) and velocity_y > 0:
	velocity_y -= 1

	# check ball bounce on border
```

```
        direction_x = bounce(pos_x,direction_x,outline_x + ball_size + velocity_x,outline_x + ou-
tline_w - ball_size - velocity_x)
        direction_y = bounce(pos_y,direction_y,outline_y + ball_size + velocity_y,outline_y + ou-
tline_h - ball_size - velocity_y)

        # check ball bounce on paddle
        #direction_x = bounce(pos_x,direction_x,outline_x + ball_size + velocity_x,pos_x + ou-
tline_w - ball_size - velocity_x)

        if pos_paddle_x < pos_x < pos_paddle_x + paddle_w and pos_y + ball_size >= pos_paddle_y:
        direction_y = bounce(pos_y,direction_y,0,pos_paddle_y - ball_size - velocity_y)

        pos_x,pos_y = move(pos_x,pos_y,velocity_x,velocity_y,direction_x,direction_y)

        draw_border()
        draw_paddle(pos_paddle_x)
        draw_ball(pos_x,pos_y)

        djb.display.show_and_wait(True) # show screen and wait...
```

Etape 12 : Rebond sur la raquette

Ajout de la détection de la sortie de balle

```
pos_paddle_x = 80
pos_paddle_y = 118
paddle_w = 16
paddle_h = 8
velocity_paddle = 2


pos_x = 80
pos_y = 64
ball_size = 3 # ball radius
velocity_x = 0
velocity_y = 2
direction_x = 1
direction_y = 1


outline_x = 0
outline_y = 20
outline_w = 120
outline_h = 108


def draw_border():
        djb.display.rect(outline_x, outline_y, outline_w, outline_h, djb.display.BLUE_H) # draw
border

def draw_ball(x,y):
        djb.display.ellipse(x, y, ball_size, ball_size, djb.display.RED_H, True) # draw ball

def draw_paddle(x):
        djb.display.rect(x, pos_paddle_y, paddle_w, paddle_h, djb.display.GREEN_H, True)


def move(p_x,p_y,vel_x,vel_y,dir_x,dir_y):
        p_x += vel_x * dir_x
        p_y += vel_y * dir_y
        return p_x,p_y


def bounce(pos,dir,border_low,border_high): # rebond de la balle
        if pos <= border_low and dir == -1: # rebond gauche ou haut
        dir = 1
        elif pos >= border_high and dir == 1: # rebond droit ou bas
        dir = -1
        return dir

while True:
        djb.scan_jst_btn() #scan buttons

        if djb.pressed(djb.btn_Right) and pos_paddle_x < outline_x + outline_w - paddle_w: # if
right button pressed
        pos_paddle_x += velocity_paddle
        if djb.pressed(djb.btn_Left) and pos_paddle_x > 0: # if left button pressed
        pos_paddle_x -= velocity_paddle

        # check ball bounce on border
        direction_x = bounce(pos_x,direction_x,outline_x + ball_size + velocity_x,outline_x + ou-
tline_w - ball_size - velocity_x)
        direction_y = bounce(pos_y,direction_y,outline_y + ball_size + velocity_y,outline_y + ou-
tline_h - ball_size - velocity_y)
```

```
# check paddle bounce
if pos_paddle_x < pos_x < pos_paddle_x + paddle_w and pos_y + ball_size >= pos_paddle_y:
direction_y = bounce(pos_y,direction_y,0,pos_paddle_y - ball_size - velocity_y)
if pos_paddle_x < pos_x < pos_paddle_x + (paddle_w//3):
        velocity_x = 2
        direction_x = -1
if pos_paddle_x + (paddle_w//3) < pos_x < pos_paddle_x + 2 * (paddle_w//3):
        velocity_x = 0
if pos_paddle_x + 2 * (paddle_w//3) < pos_x < pos_paddle_x + paddle_w:
        velocity_x = 2
        direction_x = 1

# check if losing
if pos_y + ball_size + velocity_y >= outline_y + outline_h:
djb.display.text('Game Over',40,64,djb.display.RED_H)
djb.display.show()
break

pos_x,pos_y = move(pos_x,pos_y,velocity_x,velocity_y,direction_x,direction_y)

draw_border()
draw_paddle(pos_paddle_x)
draw_ball(pos_x,pos_y)

djb.display.show_and_wait(True) # show screen and wait...
```

Etape 13 : Ajout cible et score

Ajout de d'une fonction de collision de la balle avec la cible

```
import math
score = 0

pos_paddle_x = 80
pos_paddle_y = 118
paddle_w = 16
paddle_h = 8
speed_paddle = 2

pos_x = 80
pos_y = 64
ball_size = 3 # ball radius
speed_x = 0
speed_y = 2
dir_x = 1
dir_y = 1

outline_x = 0
outline_y = 20
outline_w = 120
outline_h = 108

def draw_border():
    djb.display.rect(outline_x, outline_y, outline_w, outline_h, djb.display.BLUE_H) # draw bor-
der

def draw_ball(x,y):
    djb.display.ellipse(x, y, ball_size, ball_size, djb.display.RED_H, True) # draw ball

def draw_paddle(x):
    djb.display.rect(x, pos_paddle_y, paddle_w, paddle_h, djb.display.GREEN_H, True)

def draw_target(x,y):
    djb.display.ellipse(x, y, 10, 10, djb.display.MAGENTA_H, True)

def draw_score():
    djb.display.text('Score:', 10, 0, djb.display.WHITE_H)
    djb.display.text(str(score), 60, 0, djb.display.WHITE_H)

def move(x,y,s_x,s_y,d_x,d_y):
    x += s_x * d_x
    y += s_y * d_y
    return x,y

def bounce(v,d_v,b_low_v,b_high_v):
    if v <= b_low_v:
        d_v = -d_v
    elif v >= b_high_v:
        d_v = -d_v
    return d_v

def collide(x,y):
    distance = int(math.sqrt((x - (outline_x + outline_w)//2) ** 2 + (y - 40) ** 2))
    print(distance)
    if distance < 10:
```

```
            return distance
        return 0

while True:
    djb.scan_jst_btn() #scan buttons

    if djb.pressed(djb.btn_Right) and pos_paddle_x < outline_x + outline_w - paddle_w: # if right
button pressed
        pos_paddle_x += speed_paddle
    if djb.pressed(djb.btn_Left) and pos_paddle_x > 0: # if left button pressed
        pos_paddle_x -= speed_paddle

    # check border bounce
    dir_x = bounce(pos_x,dir_x,outline_x + ball_size + speed_x,outline_x + outline_w - ball_size
- speed_x)
    dir_y = bounce(pos_y,dir_y,outline_y + ball_size + speed_y,outline_y + outline_h - ball_size
- speed_y)

    # check paddle bounce
    if pos_paddle_x < pos_x < pos_paddle_x + paddle_w and pos_y + ball_size >= pos_paddle_y:
        dir_y = bounce(pos_y,dir_y,0,pos_paddle_y - ball_size - speed_y)
        if pos_paddle_x < pos_x < pos_paddle_x + (paddle_w//3):
            speed_x = 2
            dir_x = -1
        if pos_paddle_x + (paddle_w//3) < pos_x < pos_paddle_x + 2 * (paddle_w//3):
            speed_x = 0
        if pos_paddle_x + 2 * (paddle_w//3) < pos_x < pos_paddle_x + paddle_w:
            speed_x = 2
            dir_x = 1

    # check if losing
    if pos_y + ball_size + speed_y >= outline_y + outline_h:
        djb.display.text('Game Over',40,64,djb.display.RED_H)
        djb.display.show()
        break

    # check collide  with target
    score += collide(pos_x,pos_y)

    pos_x,pos_y = move(pos_x,pos_y,speed_x,speed_y,dir_x,dir_y)

    draw_border()
    draw_paddle(pos_paddle_x)
    draw_target((outline_x + outline_w)//2,40)
    draw_ball(pos_x,pos_y)
    draw_score()

    djb.display.show_and_wait(True) # show screen and wait...
```

Infos générales

Site web : www.coderdojobelgium.be - Contact : info@coderdojobelgium.be

ABOVE ALL, BE COOL!