2 - DojoBoy – Flappy Bird 

Cet atelier a pour objectif de coder le jeu **Flappy Bird** en MicroPython (Python) sur la console DojoBoy.

Pour comprendre les étapes de cet atelier, il est préférable de maitriser les points aborder dans l'atelier DojoBoy N°1.

Le code dans ces pages n'est pas complet. Il est indispensable de suivre les étapes l'une après l'autre pour 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

Comme pour tous les programmes destinés au DojoBoy, il faut commencer par importer le module **`DojoBoy.`**

```
from dojoboy_v1 import DojoBoy
```

Puis instancier un objet « DojoBoy » avec la variable **`djb`** .

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

Le paramètre **`False`** va permettre d'éviter d'afficher l'écran de démarrage de la console.

Maintenant, commençons à coder le jeu ! 👍

Remarque : Il n'est pas nécessaire d'écrire le texte après le caractère **#** 😉

Etape 1 : Flappy Bird = Rectangle...

Après avoir instancié la classe **DojoBoy** avec la variable **djb**, commençons par définir les bases du programme.

Pour cette première étape **Flappy Bird** prendra la forme d'un simple rectangle que nous allons déplacer verticalement avec le joystick.

Ecris le code suivant après **djb = DojoBoy(False)**

```
pos_x = 50
pos_y = 50
velocity_y = 2    # vitesse de deplacement
```

Ces variables sont utilisées pour définir la position de **Flappy Bird** à l'écran ainsi que le nombre de pixels verticaux dont se déplacera **Flappy Bird** lors de l'utilisation du joystick.

```
while True:
    djb.display.fill(djb.display.BLACK) # remplir l'ecran en noir
    djb.scan_jst_btn() # scan si joystick ou boutons appuyés

    if djb.pressed(djb.btn_Up):  # si joystick haut
        pos_y = pos_y - velocity_y  # reduit la valeur de pos_y
    if djb.pressed(djb.btn_Down):  # si joystick bas
        pos_y = pos_y + velocity_y  # augmente la valeur de pos_y

    djb.display.rect(pos_x, int(pos_y), 19, 14, djb.display.YELLOW_H, True) # dessine un rectangle a la position pos_x , pos_y
    djb.display.show_and_wait() # affiche l'ecran a 30 image par seconde
```

Le bloc de code commençant par la commande **while** défini la boucle principale du programme.

La fonction de chacune des lignes est décrite dans le commentaire après le caractère #.

Challenge

- Change la vitesse de déplacement de **Flappy Bird**
- Change le déplacement vertical par un déplacement horizontal

Etape 2 : La chute de Flappy Bird

Avec le code de l'étape 1, **Flappy Bird** ne vole pas et n'est pas attiré par le sol. Il faut corriger ce comportement pour que lors de l'appui du bouton A, **Flappy Bird** se déplace vers le haut. En cas d'arrêt de l'appuis sur le bouton A, **Flappy Bird** tombera vers le sol.

Commençons par ajouter de nouvelles variables qui vont définir la vitesse de déplacement vers le haut et l'accélération (l'augmentation de la vitesse) lors de la chute.

Remarque : Ces variables ne changeront pas lors de l'exécution du programme. Ce sont des « constantes ». Pour les différencier des variables nous les définissons avec des majuscules 😉

```
VELOCITY_Y_UP = -2.4  # vers le haut il faut "reduire" la position Y
ACCELERATION_Y = 0.2  # difference de Vitesse a chaque execution de la boucle
```

Il faut ensuite modifier la gestion des boutons avec le code suivant dans la boucle principale.

```
    if djb.pressed(djb.btn_A):  # si bouton A appuyé
        velocity_y = VELOCITY_Y_UP
        pos_y = pos_y + velocity_y  # reduit la valeur de pos_y
    velocity_y = velocity_y + ACCELERATION_Y #modifie la vitesse
    pos_y = pos_y + velocity_y # augmente la valeur de pos_y
```

Il n'est maintenant plus nécessaire de gérer les directions « haut » et « bas » du joystick (il n'y a plus que le bouton A pour faire monter **Flappy Bird**). Le code pour la direction « bas » doit donc être remplacé par les commandes pour augmenter la vitesse de chute et déplacer vers le bas **Flappy Bird.**

Le reste du code est identique.

Challenge

- Modifie la vitesse de déplacement vers le haut et examine le nouveau comportement
- Modifie la vitesse de la chute et examine le nouveau comportement
- Ajoute le code pour éviter que **Flappy Bird** disparaisse sous le sol.

Etape 3 : Les obstacles arrivent...

Flappy Bird est maintenant capable de voler.... Il est maintenant temps d'ajouter des obstacles... 😁

L'obstacle (tuyau) prendra la forme d'un rectangle qui se déplace vers **Flappy Bird**.

Il faut donc ajouter des variables pour définir la position du tuyau (« pipe » en anglais) et sa vitesse de déplacement vers **Flappy Bird**.

```
VELOCITY_PIPE_X = -2 # vitesse de déplacement vers la gauche

pipe_pos_x = djb.display.width # place le tuyau au dela du bord droit de l'ecran

pipe_pos_y = 80 # le dessus du tuyau est à la position 80
```

Il faut ensuite faire bouger le tuyau vers la gauche, l'afficher à l'écran puis lorsqu'il arrive au-delà du bord gauche le faire disparaitre.

```
    pipe_pos_x = pipe_pos_x + VELOCITY_PIPE_X # déplace le tuyau vers la gauche

    if pipe_pos_x < -26: # si le tuyau est au-dela du bord gauche (0)

        pipe_pos_x = djb.display.width #fais le apparaitre de nouveau au -dela du bord droit

    djb.display.rect(pipe_pos_x,pipe_pos_y, 26, 100, djb.display.RED_H, True)
```

Solution : Vérifie si ton code pour éviter la chute de **Flappy Bird** au-delà du sol correspond à celui-ci :

```
    if pos_y > djb.display.height - 14:

        pos_y = djb.display.height - 14
```

Challenge

- Modifie les valeurs **-14** et **-26** dans le code. Examine le comportement avec ces changements.
- Modifie la vitesse de déplacement du tuyau.
- Modifie la hauteur du tuyau

Etape 4 : En haut...en bas...

Le jeu est un peu monotone avec toujours la même hauteur de tuyau (et puis on peut toujours passer à travers le tuyau 😉). Ajoutons le code pour définir aléatoirement la hauteur du tuyau.

Comme tu as pu le vérifier, la hauteur du tuyau dépend de la valeur de la variable **pipe_pos_y**

Il faudrait donc que lorsque le tuyau disparait à gauche de l'écran, on définisse aléatoirement sa hauteur. Pour cela, il est nécessaire d'ajouter des commandes en ajoutant un module avec la ligne de code suivante en début de programme :

```
from random import randint
```

Il faut ensuite modifier le code qui fait ré apparaitre le tuyau sur le côté droit.

```
    if pipe_pos_x < -26:

        pipe_pos_x = djb.display.width

        pipe_pos_y = randint(30, djb.display.height)
```

La commande **randint** permet de définir un nombre au hasard entre deux nombres (ici **30** et le bord du bas de l'écran **128**). Ce nombre est ensuite assignée à la variable **pipe_pos_y**.

Challenge

- Modifie l'intervalle du nombre tiré au hasard pour la hauteur du tuyau. Examine le comportement.

Etape 5 : Ouch… c'est dur ce truc…

Jusqu'à présent **Flappy Bird** pouvait voler sans obstacle en passant au travers du tuyau. Il est temps de corriger la chose…

Nous allons ajouter une méthode qui va vérifier si **Flappy Bird** rentre en contact avec le tuyau. Sur base des coordonnées de position **x** et **y** et des dimensions **w** (largeur) et **h** (hauteur) de **Flappy Bird** et du tuyau, la méthode va retourner la valeur **True** s'ils se touchent ou **False** dans le cas contraire.

Le code de la méthode doit être placé juste après la définition de la variable **djb**.

```
def check_collision(x1,y1,w1,h1,x2,y2,w2,h2):
    return x1 + w1 >= x2 and x1 <= x2 + w2 and y1 + h1 >= y2 and y1 <= y2 + h2
```

Après la définition de la méthode **check_collision** , il faut maintenant l'utiliser dans note programme.

Pour l'utiliser, nous allons d'abord définir une nouvelle variable (à placer avec les autres) qui va être utilisée pour déterminer si le jeu continue ou s'arrête lorsque **Flappy Bird** touche le tuyau.

```
game_over = False
```

Ajoute le code suivant dans la boucle principale pour vérifier s'il y a une collision.

```
    if check_collision(pos_x, pos_y, 19, 14, pipe_pos_x, pipe_pos_y, 26, 100):
        game_over = True
```

En cas de collision la variable **game_over** est mise à la valeur **True**. Il faut ensuite arrêter le jeu avec le code suivant que l'on placera à la fin du code de la boucle principale.

```
    if game_over:
        djb.display.center_text("GAME OVER",djb.display.WHITE_H)
        djb.display.show()
        break
```

La commande **break** va permettre de sortir de la boucle principale et d'arrêter le jeu.

Challenge

- Modifie les valeurs 19,14, 26, 100. Examine maintenant le comportement du programme lors de collision.

Etape 6 : Trop facile...

Flappy Bird a trop facile d'éviter l'unique tuyau. Il faut maintenant ajouter un nouveau tuyau au-dessus de celui du bas ! Avant d'ajouter le code pour cela, nous allons ajouter de nouvelles constantes et méthode pour améliorer la lecture du programme.

Ces constantes définissent la taille des tuyaux ainsi que la taille de l'espace entre le tuyau du et du bas.

```
PIPE_HOLE_SIZE = 40

PIPE_W = 26

PIPE_H = 100
```

Pour permettre l'ajout du nouveau tuyau, adapte la valeur de la variable

```
pipe_pos_y = -80
```

Pour simplifier l'affichage du nouveau tuyau, ajoute la méthode **draw_pipe** après la méthode **check_collision** .

```
def draw_pipe(p_x, p_y):

    djb.display.rect(p_x,p_y, PIPE_W, PIPE_H, djb.display.RED_H, True)

    djb.display.rect(p_x,p_y + PIPE_H + PIPE_HOLE_SIZE, PIPE_W, PIPE_H,
djb.display.RED_L, True)
```

Il faut ensuite supprimer du programme le code qui permettait précédemment l'affichage de l'unique tuyau pour le remplacer par :

```
    draw_pipe(pipe_pos_x, pipe_pos_y)
```

La vérification de la collision avec le tuyau doit aussi être modifiée avec ce code :

```
    if check_collision(pos_x, pos_y, 19, 14, pipe_pos_x, pipe_pos_y, PIPE_W,
PIPE_H):

        game_over = True

    if check_collision(pos_x, pos_y, 19, 14, pipe_pos_x, pipe_pos_y + PIPE_H +
PIPE_HOLE_SIZE, PIPE_W, PIPE_H):

        game_over = True
```

Challenge

- Adapte le code pour diminuer ou augmenter la taille de l'espace verticale entre les tuyaux.
- Adapte le code pour diminuer ou augmenter la vitesse de déplacement des tuyaux.

Etape 7 : Encore trop facile...

Une seule paire de tuyau, c'est encore trop facile.... Ajoutons une nouvelle paire derrière la première.

Pour éviter la répétition de lignes de code identiques, nous allons encore ajouter de nouvelles méthodes.

La première va permettre de définir facilement la position d'une nouvelle paire de tuyau.

```
def create_pipe():
    pos_x_p = djb.display.width
    pos_y_p = randint(-PIPE_H,-PIPE_H + djb.display.height//2)
    return pos_x_p, pos_y_p
```

La seconde simplifiera la détection de collision avec les tuyaux :

```
def check_pipe_collision(p_x, p_y, p_p_x, p_p_y):
    g_over = False
    if check_collision(p_x, p_y, 19, 14, p_p_x, p_p_y, PIPE_W, PIPE_H):
        g_over = True
    if check_collision(p_x, p_y, 19, 14, p_p_x, p_p_y + PIPE_H +
PIPE_HOLE_SIZE, PIPE_W, PIPE_H):
        g_over = True
    return g_over
```

Elles sont à placer dans le programme avec les autres.

Pour gérer la nouvelle paire de tuyau, la définition des variables de position doivent être remplacées par celles-ci :

```
pipe_1_pos_x, pipe_1_pos_y = create_pipe()
pipe_2_pos_x, pipe_2_pos_y = create_pipe()
pipe_2_pos_x = pipe_2_pos_x + (djb.display.width + 19)//2
```

Dans la boucle principale, les ligne de code suivantes doivent être remplacée :

```
    pipe_pos_x = pipe_pos_x + VELOCITY_PIPE_X
    if pipe_pos_x < -26:
        pipe_pos_x = djb.display.width
        pipe_pos_y = randint(-PIPE_H,-PIPE_H + djb.display.height//2)
    if check_collision(pos_x, pos_y, 19, 14, pipe_pos_x, pipe_pos_y, PIPE_W,
PIPE_H):
        game_over = True
```

```
    if check_collision(pos_x, pos_y, 19, 14, pipe_pos_x, pipe_pos_y + PIPE_H +
PIPE_HOLE_SIZE, PIPE_W, PIPE_H):
        game_over = True
    draw_pipe(pipe_pos_x, pipe_pos_y)
```

Par ces lignes de code :

```
    pipe_1_pos_x = pipe_1_pos_x + VELOCITY_PIPE_X
    pipe_2_pos_x = pipe_2_pos_x + VELOCITY_PIPE_X
    if pipe_1_pos_x < -PIPE_W:
        pipe_1_pos_x, pipe_1_pos_y = create_pipe()
    if pipe_2_pos_x < -PIPE_W:
        pipe_2_pos_x, pipe_2_pos_y = create_pipe()
    draw_pipe(pipe_1_pos_x, pipe_1_pos_y)
    draw_pipe(pipe_2_pos_x, pipe_2_pos_y)
    if check_pipe_collision(pos_x, pos_y, pipe_1_pos_x, pipe_1_pos_y):
        game_over = True
    if check_pipe_collision(pos_x, pos_y, pipe_2_pos_x, pipe_2_pos_y):
        game_over = True
```

Challenge

- Fais une pause et amuse toi un peu 😉

Etape 8 : Flappy Bird met son costume...

Nous avons la base du jeu qui fonctionne mais le jeu est laid ! Il est temps de mettre de la couleur et de remplacer le bloc jaune qui représente **Flappy Bird** par une belle image...

Les images qui sont affichées dans les jeux d'appellent des **sprites**. Les sprites sont des images que l'on peut créer à partir d'images comme celles-ci.




Pour transformer **Flappy Bird**, ajoute ces lignes de code après la définition de la variable **djb**.

```
djb.display.add_sprite_from_file("flappy1.bin", 19, 14)
ALPHA_COLOR = djb.display.color(0,0,0)
```

Ces lignes ajoutent un sprite à l'image de **Flappy Bird** sur base du fichier « flappy1.bin » qui est doit déjà être présent dans la racine de la console.

Pour remplacer l'affichage du rectangle jaune, ajoute la ligne

```
djb.display.sprite(0, int(pos_x), int(pos_y), ALPHA_COLOR)
```

et supprime

```
djb.display.rect(pos_x, int(pos_y), 19, 14, djb.display.YELLOW_H, True)
```

Etape 9 : On ajoute de beaux tuyaux...

Pour terminer, ajoutons des sprites pour les tuyaux.

Ajoute ces lignes en dessous de celle ajoutée pour **Flappy Bird**

```
djb.display.add_sprite_from_file("pipedown.bin", 26, 100)

djb.display.add_sprite_from_file("pipeup.bin", 26, 100)
```

Il faut ensuite modifier la méthode qui affiche les tuyaux **`draw_pipe`** en supprimant les lignes qui affichent les rectangles avec celles-ci :

```
djb.display.sprite(1, p_x, p_y, ALPHA_COLOR)

djb.display.sprite(2, p_x, p_y+ PIPE_H + PIPE_HOLE_SIZE, ALPHA_COLOR)
```

Le jeu est maintenant complet mais il peut encore être amélioré...

Challenge

- Change le fond de l'image noir avec du bleu ciel (**`djb.display.color(112,197,206)`**)
- Ajoute un score ou les points sont obtenus après le passage d'une paire de tuyau par **Flappy Bird**.
- Ajout un sol qui défilent (**SPRITE**)

- Ajoute des objets bonus (**SPRITE**)

Infos générales

Auteur : CoderdojoBelgium - Date de publication : 01-2024 - CC-BY-NC-SA

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

ABOVE ALL, BE COOL!