# boo-climb

CoderDojo Ans

Décembre 2018

# Table des matières

# 1 Mise en place de l'environnement

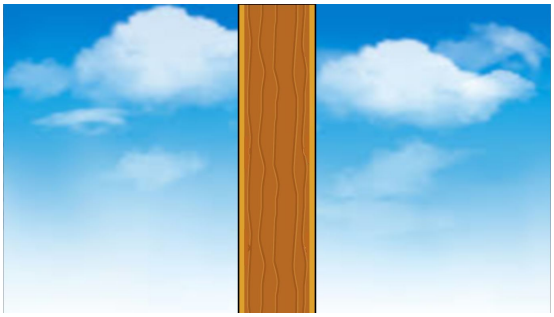
**Résumé**

Dans cette section, nous allons mettre en place l'environnement et le décor du jeu.

## 1.1 Le canevas de départ

```
# Si vous utilisez mu editor, commentez
# la ligne 'import pgzrun'
#import pgzrun

TITLE  = 'Boo climb'
WIDTH  = 1366
HEIGHT = 768

# Met à jour les objects
def update() :
    return

# Dessine les objects
def draw():

    screen.blit('sky', (0, 0))
    tree.draw()

    return

######################
## Initialisation ##
######################

# On créé et on positionne l'arbre
tree     = Actor('tree', anchor=('left', 'top'))
tree.pos = ((WIDTH - tree.width) / 2, 0)

# Si vous utilisez mu editor, commentez
# la ligne 'pgzrun.go()'
#pgzrun.go()
```

Ce qui devrait donner la situation suivante



## 1.2 Placer les branches et le boo

Dans le répertoire "images" fourni, on peut trouver plusieurs types de branches, un personnage (boo), ainsi qu'une bombe. Dans un premier temps, pour faire plus simple, nous n'allons utiliser que les branches en bon état et le personnage tel que ci-dessous.

### *Astuces*

Afin d'obtenir un bon rendu visuel et de faciliter les animations futures, il est important que les branches soit espacées de manière équidistantes! Le plus simple consiste donc à en placer une de chaque côté et de créer les autres via une boucle. Pour utiliser efficacement les boucles, les listes sont généralement d'une aide précieuse. Le code d'exemple ci-dessous montre comment les utiliser.

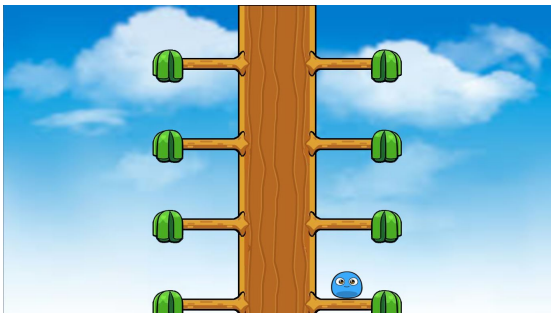
```
# Defini une liste
Mylist = []

# Ajoute un élément à la liste
Mylist.append(Actor(...))

# Accède au 3ième élément de la liste (1er = 0)
Mylist[2].pos = x, y

# J'itère toute la liste
for Element in Mylist :
  Element.x = valeur

# J'itère du 2ième au 5ième élément
for i in range(1, 4) :
  Mylist[i].y = valeur
```

### A vous de jouer

1. Placer de chaque côté de l'arbre 4 branches à droite et à gauche.
2. Positionner le personnage sur la branche la plus basse à droite.

### Vérification

A la fin de cette étape vous devriez avoir le résultat suivant.



# 2 Les animations

**Résumé**

Dans cette section, nous allons faire les animations du personnage et des branches.

Le but est de faire en sorte que le boo saute de branche en branche. Pour ce faire, à chaque fois que l'utilisateur déclenche une action, nous allons déplacer le boo latéralement au besoin et les branches verticalement.

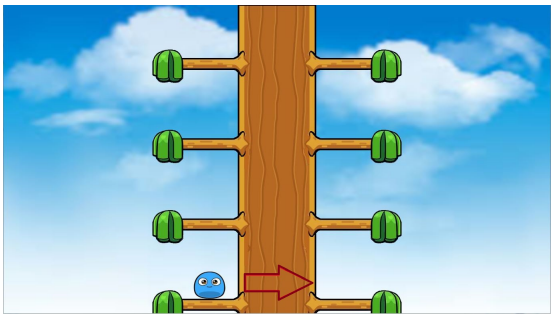
## 2.1 Déplacer le boo

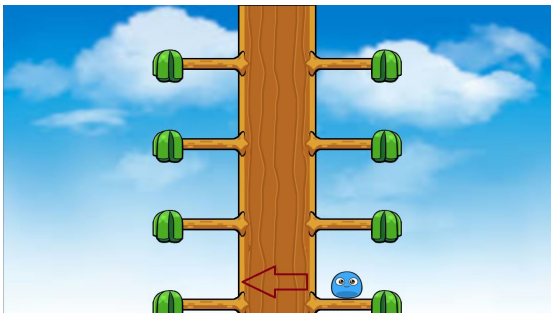
Le jeu ne comporte que 2 touches : $\leftarrow$ et $\rightarrow$.

Pour faire simple, nous allons uniquement le déplacer latéralement (i.e. seule la coordonnée x change). Il n'y a donc que 4 combinaisons possibles :

1. Le boo est à gauche et on veut aller à gauche
2. Le boo est à gauche et on veut aller à droite
3. Le boo est à droite et on veut aller à gauche
4. Le boo est à droite et on veut aller à droite

Par conséquent, il n'y a réellement que 2 combinaisons (2 et 3) où le boo doit bouger qui sont représentées ci-dessous !

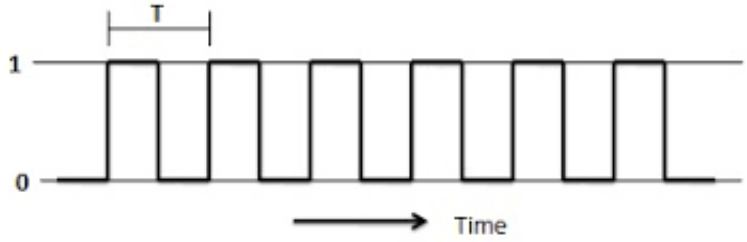




***Astuces***

1. Si X1 désigne la position de départ du boo et X2 la position d'arrivée, on peut calculer le déplacement à effectuer comme étant X = X2 - X1

2. Une fois qu'on connait cet écart X, il faut pouvoir déplacer le personnage progressivement pour donner l'impression d'un mouvement fluide. Pour ce faire, la fonction ***update()*** est toute indiquée. Comme l'indique la documentation, celle-ci est appelée 60 fois par seconde. Comme on peut le voir sur le schéma ci-dessous, on peut donc calculer le temps T qui s'écoule entre 2 appels successifs :




$$T = \frac{1}{60} = 0.01666s = 16,6ms$$

   Pour donner un exemple chiffré, si X = 60 pixels et qu'on déplace le boo d'un pixel à la fois par fonction update (e.g. boo.x += 1), il faudra 1s au boo pour atteindre sa destination.

3. Finalement, il reste à pouvoir déterminer quand l'animation commence et quand elle s'arrête pour savoir quand exécuter du code dans la fonction update et quand ne pas le faire.

   En mettant tout ça en musique, on obtient à peu près le canevas de départ suivant :

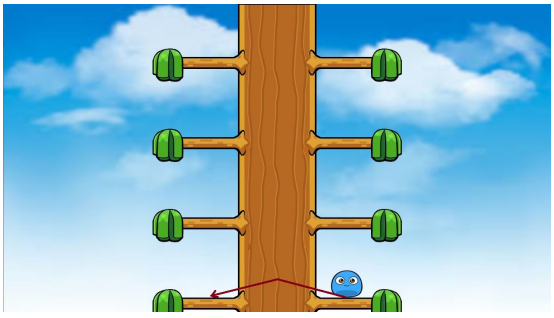
```
def on_key_down(key):
  # A gauche
  if key == keys.LEFT :
    # Où le boo est-il ?
    boo.counter = durée_du_déplacement
    boo.moveX   = ?
  # A droite
  elif key == keys.RIGHT :
    # Où le boo est-il ?
    boo.counter = durée_du_déplacement
    boo.moveX   = ?

# Apellée 60 fois par seconde
def update() :

  # Une animation est en cours
  if boo.counter > 0 :
    boo.counter -= 1

    # Déplacer le boo ici
    boo.x += boo.moveX
```

### A vous de jouer

En vous inspirant du code ci-dessus et sur base des astuces, programmer le déplacement du boo sur la gauche ou sur la droite pour les 2 touches.

#### Challenge optionnel

Pour les plus courageux d'entre vous, le challenge consiste à faire effectuer un léger déplacement vertical aller/retour (sur la composante Y) au boo durant son déplacement latéral comme illustré ci-dessous. Cela donnera un plus bel effet de saut.



### Vérification

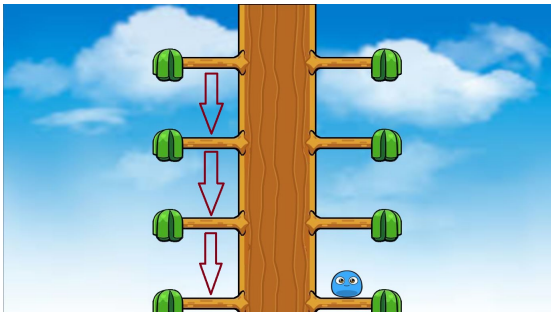
A la fin de cette étape, le boo devrait pouvoir se déplacer de gauche à droite suivant sa position et les touches sur lesquelles on appuye.

## 2.2 Déplacer les branches

Notre boo bouge mais puisque les branches sont toujours immobiles, on a pas l'impression qu'il saute réellement. Nous allons donc y remédier en faisant bouger les branches aussi.

Le principe reste le même, lorsque l'utilisateur appuye sur une de 2 touches citées précédemment, on déclenche l'animation des branches. Pour ce faire, chacune de celles-ci va descendre pour finalement atteindre la position qu'occupait précédemment la branche du dessous.

La logique est donc exactement la même que celle qu'on a employée pour faire se déplacer le boo. La différence c'est que, cette fois, seule la composante Y importe et qu'il y a plus qu'une branche.



***Astuces***

1. A mesure que les branches descendent, il doit en apparaître une nouvelle dans le dessus de l'écran. Si le code a été bien structuré avec des boucles dès le départ, ajouter une branche supplémentaire qui se trouve initialement en dehors de l'écran ne devrait pas poser de problème.

2. Une fois que les branches sont descendues et ont terminé leur animation, chaque branche s'est déplacée d'une position vers le bas. En terme algorithmique, si on considère que la branche la plus haute est la branche 0, la suivante la branche 1, etc, ... On peut considérer qu'on arrive à la situation suivante :

| | Avant | Après |
|---|---|---|
| 0 | B0 | Nouvelle branche |
| 1 | B1 | B0 |
| 2 | B2 | B1 |
| 3 | B3 | B2 |
| 4 | B4 | B3 |

Il faut donc repositionner chaque élément dans la liste et le tour est joué !

**A vous de jouer**

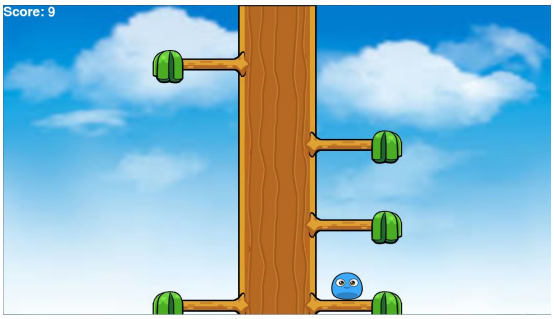
Ecrivez le code pour déplacer les branches en vous inspirant de ce que vous avez fait pour déplacer le boo.

**Vérification**

Vous devriez maintenant avoir un personnage qui, en fonction des touches enfoncées, saute de branche en branche dans l'arbre. Le déplacement cumulé du personnage et des branches doit donner l'illusion que le boo monte bel et bien dans l'arbre.

## 2.3 Faire disparaître certaines branches

Jusqu'à présent nous avons toujours afficher toutes les branches de chaque côté de l'arbre. Cela nous à permis de nous assurer que les animations fonctionnaient correctement. Néamoins, pour que le jeu présente un intérêt, il est important qu'il y ait parfois des branches manquantes de certains côtés sans quoi il est tout simplement impossible de perdre !




***Astuces***

1. Il ne peut jamais y avoir 2 branches manquantes sur le même étage, sans quoi il est impossible d'aller plus haut ! Il n'y a donc au total que 3 possibilités (les 2 branches sont là, seule la branche de gauche est là, seule

la branche de droite est là)

2. Pour faire "disparaître" un élément, le plus simple est tout simplement de ne pas le dessiner. Il est également possible en Python de dire qu'un élément n'existe pas. On peut par conséquent vérifier si l'élément existe avant d'effectuer une action sur ce dernier.

```
# Cette variable contient l'image 'tree'
variable1 = Actor('tree', anchor=('left', 'top'))

# Cette variable ne contient rien
variable2 = None

def draw() :

  # Ce test est vrai, variable1 contient quelque chose
  if variable1 :
    variable1.draw()

  # Ce test est faux, variable2 ne contient rien
  if variable2 :
    variable2.draw()
```

3. Finalement, pour que le jeu soit plaisant et différent à chaque fois, il est important de faire apparaître aléatoirement les branches. L'exemple suivant montre comment générer un nombre aléatoire en Python

```
# On importe et initialise le module
import random

random.seed()

# On récupère aléatoirement un chiffre entre 1 et 3
value = random.randint(1, 3)
```

**A vous de jouer**

Programmer la logique qui fait apparaître les nouvelles branches pour que, de manière aléatoire, il n'y ait pas toujours 2 branches de chaque côté de l'arbre.

**Vérification**

A la fin de cette étape votre boo est capable de grimper de branche en branche. A chaque fois qu'il saute de nouvelles branches apparaissent suivant un ordre imprévisible.

# 3 La logique du jeu

**Résumé**

Nous avons jusqu'à présent programmé tous les éléments visibles. Cependant, il n'est toujours pas possible à ce stade ni de gagner, ni de perdre. En effet, même si votre personnage saute du côté où il n'y a pas de branche, rien ne se passe. Il est donc à présent temps de passer à la logique opérationnelle du jeu elle-même

## 3.1 Comptabiliser les points

Comme dans tout jeu qui se respecte un compteur avec des points est toujours sympathique. Pour ce faire, on peut compter chaque fois que le boo parvient à sauter sur une branche valide.

Idéalement le score devrait être affiché à l'écran.

***Astuces***

1. Penser à bien initialiser le score à 0 en début de partie.
2. N'incrémenter le score qu'à la fin de chaque animation réussie

**A vous de jouer**

Ajouter et afficher le score du jeu.

**Vérification**

Vous devriez à présent avoir un personnage qui est capable de monter de branche en branche avec un score qui représente le nombre de saut réussi.

## 3.2 Perdre le jeu

Le jeu est perdu lorsque le boo saute d'un côté où il n'y a pas de branche. Au vu de tout ce que vous avez déjà fait, il ne devrait pas être trop compliqué de détecter cette situation.

***Astuces***

1. Le boo ne peut se trouver qu'à 2 positions qui correspondent toujours au même niveau sur l'arbre. Il suffit donc de regarder, en fin d'animation, si la branche où il se trouve est visible ou pas

2. Dans le cas où le jeu est perdu, puisqu'il n'y a plus de branche, prévoir une animation où le boo tombe jusqu'à sortir de l'écran serait du plus bel effet.

 **A vous de jouer**

Détecter que le boo a sauté d'un côté ou il n'y a plus de branche. Le faire tomber hors de l'écran et afficher un texte qui dit "Game over".

### 3.3 Redémarrer le jeu

Une fois que le jeu est perdu, l'utilisateur aura peut-être à coeur de refaire une partie. Il serait donc souhaitable qu'il puisse redémarrer le jeu immédiatement sans avoir à le quitter et le redémarrer.

 ***Astuces***

1. Choisir une touche pour faire redémarrer le jeu (exemple : ENTER ou bien n'importe quelle touche)

2. Il ne faut redémarrer le jeu que s'il est perdu

3. Tout le code est déjà là mais il faut peut-être probablement le remettre dans des fonctions pour éviter de le dupliquer

 **A vous de jouer**

Une fois le jeu perdu, permettre à l'utilisateur de redémarrer une nouvelle partie en appuyant sur une touche.

 **Vérification**

Ca y est ! Vous avez une première version d'un jeu tout à fait jouable. Félicitations !

## 4 Pour aller plus loin

### 4.1 Challenge 1 : ajouter des bombes

Les images fournies contiennent une bombe.

Pour pimenter le jeu, on pourrait aléatoirement ajouter des bombes sur certaines branches. Le jeu serait alors également perdu si le boo venait à sauter sur une branche avec une bombe.

***Astuces***

1. Il ne peut y avoir de bombes que lorsque les branches sont présentes de chaque côté de l'arbre

2. Il faut animer les bombes de la même manière que les branches.

3. Il faut détecter la présence des bombes de la manière que les branches

## 4.2 Challenge 2 : ajouter des branches qui cassent

Les images contiennent également des branches abîmées.

Ce qui pourrait rajouter encore plus de défi pour l'utilisateur c'est d'intégrer des branches abîmées. Le principe d'une branche abîmée étant qu'on peut sauter dessus mais qu'on ne peut pas y rester trop longtemps sous peine qu'elle se casse !

***Astuces***

1. La branche cassée est à priori une branche comme les autres. C'est juste l'image qui est différente. On peut donc savoir en inspectant la propriété ***image*** de l'objet s'il s'agit d'une branche cassée ou non.

2. Il faut pouvoir détecter que le boo vient de sauter sur une branche cassée et démarrer un compteur pour comptabiliser le temps qu'il reste dessus (voir 2.1 déplacer le boo).

3. Ce compteur doit être remis à zéro à chaque fois que le boo saute à un autre endroit.

4. Si le compteur dépasse la valeur d'attente, la branche disparait et se transforme en "pas de branche" : le jeu est perdu.