

Animation des entités du jeu avec Arcade

I/ Animation du joueur

A ce stade, toutes les textures du joueur ont été chargées et la gestion du clavier (via les flèches directionnelles) a été effectué. Il ne reste donc qu'à s'occuper de l'animation du joueur en fonction de son déplacement et de son statut.

L'attribut *textures* gère l'**ensemble des textures** et est une **liste de listes**. L'attribut *texture* (qui vient de la classe *'Sprite'* gère la texture à afficher). Il faudra en conséquence chercher la bonne texture à afficher.

L'attribut *current_texture_indice* a le rôle **d'indice de la liste de textures** correspondant au **statut actuel du joueur** : il ne doit pas dépasser le nombre d'images de disponible !

L'attribut *status* indique le **statut** du joueur niveau '*animation*' (WALK_LEFT, ATTACK_DOWN etc.).

Voici les 3 cas à considérer :

- **Cas 1** : Si le joueur **ne se déplace pas, pas d'animation**, il suffit de sortir de la méthode avec l'instruction *return* (Déjà écrit dans le squelette du programme).
- **Cas 2** : Si le joueur **change de direction**, l'attribut *status* du joueur doit être modifié et l'attribut *current_texture_indice* est ramené à 0.
- **Cas 3** : Si le joueur se **déplace dans la même direction**, l'attribut *current_texture_indice* est incrémenté de 1. S'il dépasse *len(self.textures[self.status])* alors il est ramené à 0 soit au début de l'animation.

Cas 1 : Si les attributs *change_x* et *change_y* sont mis à **zéro** (considérés donc comme **False**), le joueur ne bouge plus donc pas d'animation.

Cas 2 : Traite le **changement** de type de **déplacement** du joueur (suivre l'exemple).

Après les traitements, nouvelle texture à appliquer. Retour à la première si besoin.

```
# Animation
# Si le joueur ne bouge pas, pas d'animation
if not self.change_x and not self.change_y :
    return

# Si changement de type de déplacement
if self.change_x < 0 :
    self.status = PLAYER_WALK_LEFT
    self.current_texture_indice = -1
#### A COMPLETER ####

# Passage à la texture suivante (animation)
self.current_texture_indice += 1

# Image suivante sauf si l'on est à la fin de l'animation (retour au début)
if self.current_texture_indice >= len(self.textures[self.status]) :
    self.current_texture_indice = 0

# Nouvelle texture à afficher
self.texture = self.textures[self.status][self.current_texture_indice]
```

1/ **Compléter** le cas N°2. Vérifier qu'effectivement que l'affichage évolue si le joueur change de direction.

Aide : L'étude du **signe** des attributs *change_x* et *change_y* permet de déterminer le statut du joueur.

2/ (*) **Tester** en appuyant sur deux directions simultanément et en continu : **si l'animation change à chaque déplacement, cela est un bug** : proposer une amélioration de l'algorithme (Cas N°2).

Appeler le professeur pour validation.

II/ Animation des monstres

1/ Dans le fichier `mob.py`

Tout comme pour le joueur, les textures sont chargées à ce stade. Il ne reste plus qu'à régler le **déplacement aléatoire** et l'**animation des monstres**. ➡

4/ A **compléter** (astucieusement) si l'animation est trop rapide 😊.

5/ A **compléter** en comparant les attributs `len(self.textures[self.status])` et `current_texture_indice`.

6/ A **compléter** en suivant l'exemple.

7/ **Mettre à jour** les attributs `center_x` et `center_y` en fonction de `change_x` et `change_y`.

8/ A **compléter**.

Aide : on considérera les valeurs des variables `WALK_DOWN`, `WALK_LEFT`, `WALK_UP` et `WALK_RIGHT` pour utiliser `randint(a,b)`.

9/ A **compléter**.

```
def update(self) :
    # Ralentissement de la vitesse d'animation
    ##### A COMPLETER #####

    # Si l'indice courant de la texture dépasse le nombre d'images de l'animation
    # en cours, on revient à l'indice de la texture de départ.
    ##### A COMPLETER #####

    # Déplacement du monstre (jusqu'au bout du compteur initialisé)
    if self.current_count_tick_move < self.init_count_tick_move :
        if self.status == WALK_DOWN :
            self.change_y = -self.attributes["Speed"]
            ##### A COMPLETER #####

        # Déplacement du monstre
        ##### A COMPLETER #####

        # Incréméntation du compteur de mouvement
        self.current_count_tick_move += 1
    # Déplacement aléatoire et réinitialisation des attributs
    else :
        # Remise à zéro du compteur et de l'indice de la texture à afficher
        self.current_count_tick_move = 0
        self.current_texture_indice = 0

        # Nouveau type de déplacement (self.status)
        ##### A COMPLETER #####

        # Initialisation aléatoire du nombre de déplacements
        # (self.init_count_tick_move).
        ##### A COMPLETER #####

    # Réinitialisation du déplacement et nouvelle texture
    self.change_x, self.change_y = 0,0
    self.texture = self.textures[self.status][self.current_texture_indice]
```

2/ Dans le fichier `main.py`

11/ L'attribut `mobs` est la liste des entités du jeu.
Appeler la méthode `update(self)` pour chacun d'entre eux.

```
def on_update(self, delta_time):
    # Déplace le joueur, gère les collisions avec les objets de la map.

    # Mise à jour du joueur

    # Mise à jour des mobs
```

12/ **Relancer** le kernel puis **exécuter** le jeu. **Régler** ensuite la vitesse des animations (question 4/)

