

# Chargement des monstres du jeu avec *Arcade*

## I/ Préparation des monstres

### 1/ Avec le logiciel GIMP

Le principe est **identique** qu'avec le joueur.

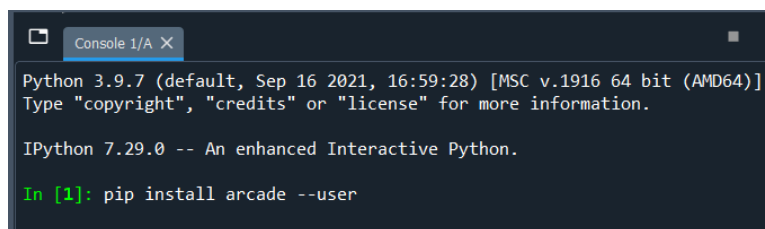
## II/ Chargement des monstres dans le jeu RPG

### 1/ Le fichier `constants.py`

Préalable nécessaire : **s'assurer** que les fichiers nécessaires sont dans le répertoire /Mobs.

1/ **Ouvrir** l'EDI *Spyder* et **charger** les fichiers « squelette ».

**Important** : dans la partie `console`, **importer** le bibliothèque *Arcade* avec l'instruction suivante :  
*pip install arcade --user*. **Appuyer** sur la touche `Entrée` pour exécuter l'instruction (cela peut prendre quelques minutes).



```
Python 3.9.7 (default, Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license()" for more information.

IPython 7.29.0 -- An enhanced Interactive Python.

In [1]: pip install arcade --user
```

2/ Dans le fichier *constants.py*, **adapter** les données au fichier *orc.png*.

```
# Caractéristiques (image) de l'orc
### A MODIFIER si autre image ###
ORC_WIDTH, ORC_HEIGHT = 32, 32
ORC_SCALING = 2
ORC_FILE = "Mobs/orc.png" # Adapter les chemins relatifs du fichier
MOB_CHARACTERISTICS_FILE = "Mobs/mob.json" # Idem

# Indice des animations des joueurs
### NE PAS CHANGER ###
WALK_DOWN, WALK_LEFT, WALK_RIGHT, WALK_UP = 0, 1, 2, 3
ATTACK_DOWN, ATTACK_LEFT, ATTACK_RIGHT, ATTACK_UP = 4, 5, 6, 7

# Coordonnées des images de chaque animation de l'orc
### A COMPLETER / MODIFIER si autre image ###
ORC_WU_COORDS = [(0,0), (32,0), (64,0), (96,0), (128,0), (160,0), (192,0), (224,0), (256,0)]
ORC_WL_COORDS = []
ORC_WD_COORDS = []
ORC_WR_COORDS = []

ORC_AU_COORDS = []
ORC_AL_COORDS = []
ORC_AD_COORDS = []
ORC_AR_COORDS = []

# Regroupement des coordonnées précédentes dans une liste
### NE PAS CHANGER ###
ORC_SPRITE_COORDS = [ORC_WD_COORDS, ORC_WL_COORDS, ORC_WR_COORDS, ORC_WU_COORDS, \
                     ORC_AD_COORDS, ORC_AL_COORDS, ORC_AR_COORDS, ORC_AU_COORDS ]
```

Remarque : mettre la constante *MAP\_SCALING* à 2. De même pour *ORC\_SCALING* (sauf si déjà fait).

## 2/ Le fichier `entity.py`

Ce fichier est déjà complété à ce stade.

## 3/ Le fichier `mob.py`

Ce fichier contient la classe `Mob` qui **hérite** de la classe `Entity`. Cette dernière s'occupe notamment du chargement des animations de l'entité (donc de celles des monstres 😊).

Toutes les informations nécessaires se trouvent à ce lien :

[https://github.com/Imayer65/NSI\\_T/blob/main/Projets/PRJ\\_RPG/Partie\\_III\\_NPC/Pr%C3%A9sentation\\_Partie\\_III.pdf](https://github.com/Imayer65/NSI_T/blob/main/Projets/PRJ_RPG/Partie_III_NPC/Pr%C3%A9sentation_Partie_III.pdf) (Page 9 et 10).

Voici le fichier *mob.json* qui regroupe les **caractéristiques** d'un orc. Il s'agit d'un dictionnaire de dictionnaires (une seule clé ici mais on peut imaginer l'ajout d'autres clés en fonction des caractéristiques d'autres monstres).

1/ **Ajouter** la clé « *Level* » et la mettre à la valeur de « 1 ».

```
{ "Orc" :  
  {  
    "Attack" : 10,  
    "Block" : 0.05,  
    "Defense" : 20,  
    "Dodger" : 0.05,  
    "HitPoints" : 20,  
    "Name" : "Orc",  
    "Parry" : 0.10,  
    "Speed" : 2  
  }  
}
```

A noter que les **positions initiales des monstres** seront **aléatoires** : les valeurs de clés *Init\_x* (en abscisse) et *Init\_y* (en ordonnées) sont inutiles dans le fichier JSON.

Les deux méthodes suivantes donnent respectivement le nom du monstre (*orc* ici) et la position initiale du monstre (qui sera à générer aléatoirement).

2/ **Compléter** les deux méthodes de la classe `Mob` comme indiqué.

```
def set_name(self, name) :  
    self.attributes['Name'] = name  
  
def set_init_position(self, x_pos, y_pos):  
    self.center_x = x_pos * MAP_SCALING  
    self.center_y = y_pos * MAP_SCALING  
    self.init_x_pos = self.center_x  
    self.init_y_pos = self.center_y
```

Ce fichier JSON est chargé dans la méthode *setup(self)* et les couples (clé, valeur) sont mises dans l'attribut *attributes* (voir à droite).

3/ Dans le fichier *constants.py*, **ajouter** le chemin relatif du fichier *mob.json* dans une variable appelée *ORC\_CHARACTERISTICS\_FILE*.

4/ **Compléter** la méthode *setup(self)* si besoin comme indiqué.

```
def setup(self) :  
    # Chargement des textures  
    super().setup()  
  
    # Ouverture du fichier JSON file  
    # Chargement des caractéristiques du joueur  
    f = open(MOB_CHARACTERISTICS_FILE)  
    data = json.load(f)  
  
    for key,value in data[self.attributes['Name']].items():  
        self.attributes[key] = value  
  
    # Fermeture du fichier  
    f.close()  
  
    self.status = 0  
    self.texture = self.textures[0][0]
```

Comme pour avec le joueur, on initialise par défaut la texture à afficher pour le monstre.

```
# Texture de départ : à la base, joueur se déplaçant vers le bas  
self.texture = self.textures[0][0]
```

Enfin, la méthode *update(self)* sera **complétée** lors de l'**activité de l'animation**.

## 4/ Le fichier `main.py`

C'est ici que l'on va créer les monstres, notamment dans la méthode `setup(self)`. C'est l'attribut ***mobs*** de la classe `My_Game`` qui est **chargée** de **stocker** tous les **monstres**.

5/ Dans la partie *# Création de X orcs placés aléatoirement, X au choix ;)*, **créer** une liste de 15 orcs.

Pour se faire :

- **Créer** une boucle adaptée.
- **Créer** une instance de la classe `Mob`` avec les arguments demandés.
- **Mettre** le nom du monstre à « Orc » à l'aide de la méthode `set_name(self, name)`.
- **Positionner** le monstre de manière « aléatoire » (\*), en lien avec votre carte (éviter qu'un monstre soit dans l'eau par exemple 😊 ) à l'aide la méthode `set_init_position(self, x_pos, y_pos)`.
- **Ajouter** l'instance dans l'attribut `mobs`.

(\*) Pour la génération du pseudo-aléatoire en Python, lien ici : <https://fr.acervolima.com/randint-fonction-en-python/>

6/ Ce code ajoute les monstres dans `sprite_list` (gestion des collisions et de l'affichage notamment). **Compléter** à l'endroit indiqué (méthode `setup(self)` ).

```
# ATTENTION : avant l'appel au setup(), plantage sinon !!
for mob in self.mobs :
    self.sprites_list.append(mob)
    mob.setup()
```

L'affichage des monstres est assuré par l'instruction `self.sprite_list.draw()` de la méthode `draw(self)` de la classe `My_Game``. Il n'y a donc rien à ajouter ici.

7/ **Redémarrer le kernel** et **vérifier** que les monstres s'affichent correctement. Là encore, ne pas hésiter à charger une autre texture que celle de base :

```
self.texture = self.textures[0][0]
```