Notre groupe est constitué de Berke, Lucas, et Ilyes. On a décidé de faire un Street Fighter II en Python avec le module Pygame. Tout d'abord nous avons d'abord mis en place le squelette du jeu c'est-à-dire faire une classe Player accompagné du constructeur et de tous ses attributs et les différentes méthodes de cette classe Player.

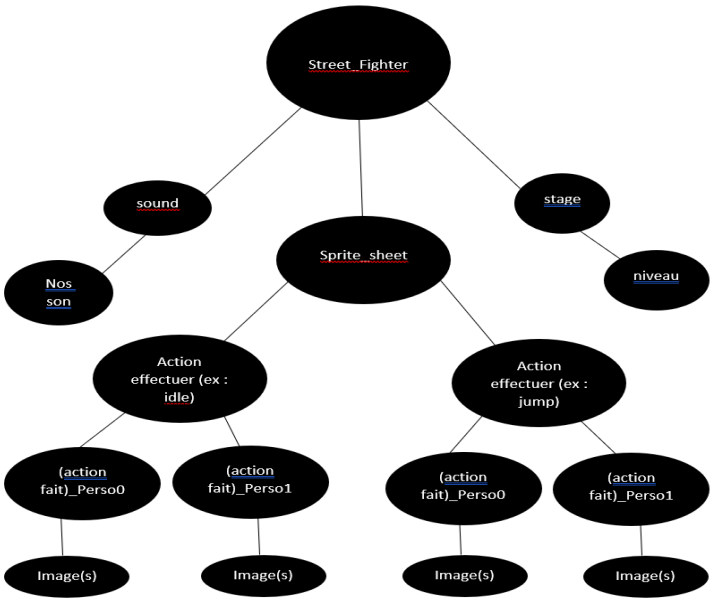
Constructeur de la Class Player :

```python
import pygame as pg
from pygame.locals import *
import os

class Player():
    """on créer une classe player avec tous les attribut d'un joueur
    x est la coordonnée x du joueuer
    y est la coordonnée y du joueuer
    force est les degats qu'il vas donner
    """
    def __init__(self, x, y, force, orientation):
        self.x = x
        self.y = y
        self.orientation = orientation # orientation du presonnage
        self.force = force # puissance de frappe de notre personnage qui varie en fonction du personnage choisie
        self.PV = 300 # les Points de Vie de notre personnage
        self.height = 350 # hauteur de notre personnage
        self.width = 200 # largeur de notre personnage
        self.isMoving = False # si le joueur bouge
        self.isPunch = False #cout de poing booléen
        self.isKick = False # coute de pied booléen
        self.isCrouch = False # accroupis booléen
        self.isJump = False # saut booléen
        self.isHadoken = False # si on lance une boule de feu
        self.isTouched = False # si on est toucher
        self.jump_height = 10 # hauteur max du saut de notre personnage
        self.sprites = {"idle": [], "punch": [], "hit": [], "jump": [], "walk": []}
        for dire in os.listdir("D:\Street Fighter\sprite_sheet"): #on charge toutes les images dont on a besoin
            for dire_ in os.listdir(f"D:\Street Fighter\sprite_sheet\{dire}"):
                for file_name in os.listdir(f"D:\Street Fighter\sprite_sheet\{dire}\{dire_}"):
                    self.sprites[str(dire)].append(pg.image.load(f"D:\Street Fighter\sprite_sheet\{dire}\{dire_}\{file_name}"))
        self.hitbox = pg.Rect(self.x, self.y, 200, 350) #on recupere la taille de notre image et on l'appelle hitbox car pygame gere les hitbox/colisions avec les Rect
        self.hitbox.x = self.x # permet de mettre les personnages aux bons endroits (car sinon (0,0))
        self.hitbox.y = self.y # permet de mettre les personnages aux bons endroits (car sinon (0,0))
        self.projectile_hb = pg.Rect(self.hitbox.right, self.hitbox.centery - 50/2, 50, 50)

```

Les arguments demander par le constructeur sont la position initiale du joueur (x,y) la puissance de frappe du Joueur (force) et son orientation (orientation) qui est « right » ou « left ». Nous avons donc tous les attributs qu'une class Playeur peut avoir comme ses dimensions (height,width) ses Points de Vie (PV) et tous les booléen qui commence par self.isXxxx (le « is »signifie que c'est un booléen True ou False). Pour stocker les Sprite sheets (les images permettant d'animer le joueur) nous avons décider de partir sur un dictionnaire dans le quelle la clé est l'action effectué et la valeur une liste de surface (nos images). Cependant tous les personnages n'ont pas les mêmes animations. Nous avons donc décidé de hiérarchisé nos dossiers contenant nos images de la manière suivante :



Cette hiérarchie nous a permis de faire une boucle qui parcourt notre dossier et qui correspond au personnage choisie (ligne 28 a 31) grâce au module OS que nous importons.

Dans le code générale nous appliquons la méthode .display sur nos instance Player. Cette méthode est faite a chaque passage de la boucle. Elle permet de changer la direction du joueur si le Player1 se retrouve derrière Player2. Elle permet aussi d'afficher la bonne image au bon moment (ex : si le personnage ne bouge pas alors l'animation d'Idle est jouer (ce n'est pas encore le cas d'où le [0] pour récupéré que la première image))

```
def display(self, win, player_rect, p2):
    """methode qui affiche le joueur sur
    win : la Surface sur le quelle on souhaite afficher notre personnage
    player_rect : le Rect (hitbox) de notre personnage
    p2 : instance Player adverse (direction)
    """
    # si le Joueur est derriere l'adversaire leurs orientation s'inverse
    pg.draw.rect(win, (255,255,255),player_rect) # on affiche la HB des joueurs
    if self.hitbox.centerx > p2.hitbox.centerx: #si derriere l'adversaire
        self.orientation = "left"
    elif self.hitbox.centerx < p2.hitbox.centerx: # si devant l'adversaire
        self.orientation = "right"

    # si on est toucher alors on affiche le perso toucher
    if self.isTouched:
        if self.orientation == "right":
            touched = pg.transform.scale(self.sprites["hit"][0], ((self.width,self.height)))# on charge l'image qui est orienté de base a droite
            win.blit(touched, self.hitbox) # on affiche
        elif self.orientation == "left":
            touched = pg.transform.scale(self.sprites["hit"][0], ((self.width,self.height)))# on charge l'image qui est orienté de base a droite
            touched = pg.transform.flip(touched, True, False)# on tourne l'image a gauche
            win.blit(touched, self.hitbox) # on affiche l'image
        self.isTouched = False # on reinitialise isTouched

    # si on ne bouge pas on affiche le perso Idle dans la HB
    elif self.isMoving == False:
        #on affiche idle en fonction de l'orientation
        idle = self.sprites["idle"][0]
        if self.orientation == "right": #si l'orientation est droite on affiche l'image qui regarde a droite
            idle = pg.transform.scale(idle, (self.width,self.height))
            win.blit(idle, self.hitbox)
        elif self.orientation == "left": #si l'orientation est gauche on affiche l'image qui regarde a gauche
            idle = pg.transform.scale(idle, (self.width,self.height))
            idle = pg.transform.flip(idle, True, False)
            win.blit(idle, self.hitbox)

    # si le perso avance on affiche le perso entrain de bouger dans la HB
    elif self.isMoving == True:
        walk = self.sprites["walk"][0]
        if self.orientation == "right":
            walk = pg.transform.scale(walk, (self.width,self.height))
            win.blit(walk, self.hitbox)
        elif self.orientation == "left":
            walk = pg.transform.scale(walk, (self.width,self.height))
            walk = pg.transform.flip(walk, True, False)
            win.blit(walk, self.hitbox)
```

Pour finir voici la méthode .puch qui soustrait self.force du Player1 a au PV de Player2 (p2.pv) et frappe a gauche ou a droite en fonction de la direction du joueur :

```
def punch(self, p2, win):
    """enleve des degats du joueur p1 au joueur p2
    si la hitbox du coup de p1 touche la hitbox du joueur p2
    p2 : joueur adversaire instance Player
    win : las surface sur le quelle on affiche le rectangle
    """
    if self.isKick == False: #si on ne fait pas deja un coup de pied
        if self.orientation == "right": # si on regarde a droite alors la hb du poing pointe a droite
            self.isPunch = True
            punch_hb = pg.Rect(self.hitbox.centerx, self.hitbox.centery - 100, 200,60)# on definie la hitbox du coup
            pg.draw.rect(win, (255,0,0), punch_hb) #on dessine pour visualiser (a enlever une fois que le personnage sera afficher)
            if punch_hb.colliderect(p2.hitbox): #si la hb du coup touche la hb du joueur adverse
                if p2.PV > 0: #si sa vie est > 0
                    p2.PV -= self.force #alors on lui enleve de la vie
                    p2.isTouched = True
        if self.orientation == "left":# si on regarde a droite alors la hb du poing pointe a gauche
            self.isPunch = True
            punch_hb = pg.Rect(self.hitbox.centerx - 200, self.hitbox.centery - 100, 200,60)
            pg.draw.rect(win, (0,0,255), punch_hb)
            if punch_hb.colliderect(p2.hitbox): #si la hb du coup touche la hb du joueur adverse
                if p2.PV > 0: #si sa vie est > 0
                    p2.PV -= self.force #alors on lui enleve de la vie
                    p2.isTouched = True

        self.isPunch = False #on remets le booléen a False
```

Enfin voici toutes les autres méthodes disponibles et certaines à compléter :

```python
def kick(self, p2, win):
    """enleve des degats du joueur p1 au joueur p2
    si la hitbox du coup de p1 touche la hitbox du joueur p2
    p2 : joueur adversaire instance Player
    win : las surface sur le quelle on affiche le rectangle
    """
    if not self.isPunch: # si on ne fait pas deja un coup de poing
        if self.orientation == "right":
            self.isKick = True
            kick_hb = pg.Rect(self.hitbox.centerx, self.hitbox.centery + (self.height//2)//2, 200, 100) #la hb du coup de pied
            pg.draw.rect(win, (255,0,0), kick_hb) # on affiche cette HB
            if kick_hb.colliderect(p2.hitbox): # si il y a collision
                if p2.PV > 0: # et que les PV du joueur toucher n'est pas nul
                    p2.PV -= self.force # on enleve des pv
                    p2.isTouched = True
        if self.orientation == "left":# si on regarde a droite alors la hb du poing pointe a gauche
            self.isKick = True
            kick_hb = pg.Rect(self.hitbox.centerx - 200, self.hitbox.centery + (self.height//2)//2, 200, 100) #la hb du coup de pied
            pg.draw.rect(win, (0,0,255), kick_hb) # on affiche cette HB
            if kick_hb.colliderect(p2.hitbox): #si la hb du coup touche la hb du joueur adverse
                if p2.PV > 0: #si sa vie est > 0
                    p2.PV -= self.force #alors on lui enleve de la vie
                    p2.isTouched = True
    p2.isTouched = False # on reintialise
    self.isKick = False # on reinitialise
```

```python
def hadoken(self, p2, win):
    if self.isPunch == False and self.isKick == False:# si on ne fait pas deja une autre action
            self.isHadoken = True
            pg.draw.rect(win, (255,0,0), self.projectile_hb) # on affiche le projectile
            if not self.projectile_hb.colliderect(p2.hitbox): # tant que on a pas toucher l'adversaire
                if self.projectile_hb.x < self.hitbox.x + 500: # tant que le projectile ne sera pas assez loin
                    self.projectile_hb.x += 10 # on fait avancer le projectile
                else: # si le projectile est assez loin
                    self.isHadoken = False # on remets le booléen en False
                    self.projectile_hb = pg.Rect(self.hitbox.right, self.hitbox.centery - 50/2, 50, 50) #on reinitialise les coordonnée du projectile
            else:
                if p2.PV > 0: #si sa vie est > 0
                    p2.PV -= self.force #alors on lui enleve de la vie
                    p2.isTouched = True
                    p2.isTouched = False
                    self.isHadoken = False # on remets le booléen en False
                    self.projectile_hb = pg.Rect(self.hitbox.right, self.hitbox.centery - 50/2, 50, 50) #on reinitialise les coordonnée du projectile
```

à optimiser et revoir

```python
def move(self, get_pressed):
    """bouge le joueur sur l'axe x,
    attend en argument le bouton appuyer get_pressed
    get_pressed : le bouton appuyer
    """

    x_vel = 10 # la velocité des joueurs
    if get_pressed[K_RIGHT]:
        self.hitbox.x += x_vel
        self.isMoving = True
    elif get_pressed[K_LEFT]:
        self.hitbox.x -= x_vel
        self.isMoving = True
    else:
        self.isMoving = False
```

```python
def crouch(self):
    """
    on reduit la hitbox du personnage et le personnage s'accroupis
    """
    ############################################################
    # faire en sort que l'action ne se produit q'une fois      #
    # meme si on reste appuyer sur fleche bas                  #
    ############################################################

    # self.hitbox.height = 350/2 #on reduit la HB
    # self.hitbox.width += 50 # on agrandis un peu la largeur de la HB
    # self.hitbox.y += 350/2 # on descend le joueuer de la moitié de la hitobx
    # self.hitbox.height = 350 #on revient a l'etat initiale
    # self.hitbox.width = 200
    # self.isCrouch = False
    pass
```

```python
def jump(self):
    """methode permettant au personnage de sauter"""
    if self.jump_height >= -10:# si on est pas a la hauteur max de notre saut
        self.hitbox.y -= (self.jump_height * abs(self.jump_height)) #on bouge le joueur (abs est pour les valeurs neg)
        self.jump_height -= 1 #on decremante
    else: #saut terminer reinitialisation de nos valeurs
        self.jump_height = 10
        self.isJump = False
```

Taches : Berke a fait la hiérarchisation et a mis en forme le squelette du jeu qui est aussi partager avec Lucas qui a aussi fait les collisions entre les joueurs et les coups etc et Ilyes a fait l'affichage du niveau en fond a la bonne taille, l'affichage des personnage dans les hitbox et le HUD (barre de vie)