LYCÉE RÉGIONAL d'Enseignement Général et Technologique Arthur VAROQUAUX	python ISN. Pygame(*les Rect()*)	Term S

Les Rect()…objet pygame pour stocker des données rectangulaires.

Ils facilitent le déplacement des personnages/objets et permettent de gérer les collisions

Création d'un Rect() :

On peut créer un objet Rect en indiquant la position x,y de son coin en haut à gauche ainsi que sa largeur et sa hauteur.

Exemple : un rectangle dont le coin en haut à gauche se situe aux coordonnées (**10,20**) et qui fait **150** de largeur et de hauteur)

nouveau_rect=Rect(10,20,150,150)

On peut également créer un Rect() à partir d'autres objets telles que des images ou des surfaces.

Exemple :

perso=pygame.image.load("perso.png")

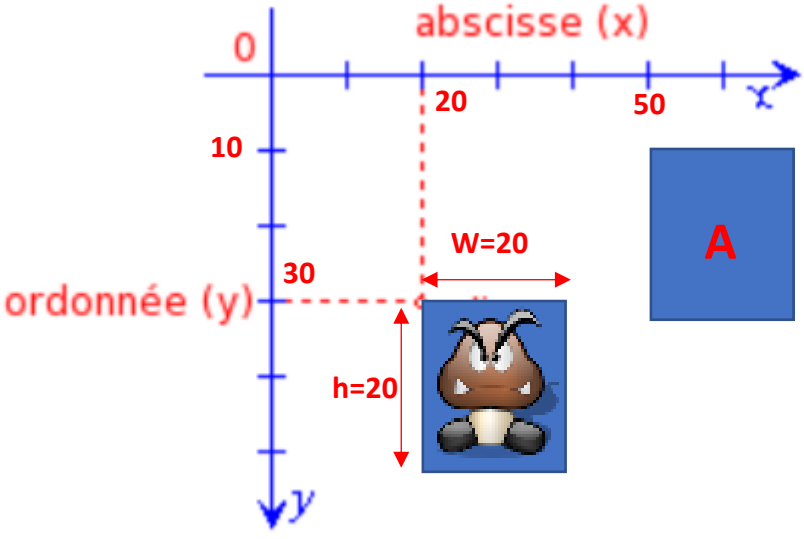
persorect=perso.get_rect()

persorect est un objet **Rect**() qui contiendra les coordonnées **x** et **y** de sa position (0 et 0 au départ) ainsi que sa **largeur** et sa **hauteur** (ici celles de l'image **perso**). Il s'agit d'un rectangle invisible « ***encadrant*** » l'image

A quoi çà sert ? et bien, une fois l'objet Rect créé, on ne travaillera plus sur l'image elle-même, mais on travaillera avec les objets Rect() qui ont des attributs permettant de les déplacer, positionner facilement.

```
x,y
top, left, bottom, right
topleft, bottomleft, topright, bottomright
midtop, midleft, midbottom, midright
center, centerx, centery
size, width, height
w,h
```

Exemple : on considère une image et son Rect() associé indiqué par un rectangle bleu.



persorect.bottom, persorect.top
persorect.left et persorect.right
Permettent d'avoir accès aux 4 bords de l'image.

persorect.x et persorect.y nous donnent les coordonnées du coin supérieur gauche.
Si on veut les modifier on peut écrire :

persorect.x=50
persorect.y=10

et le rectangle bleu se positionnera à une nouvelle position(ici en A)

Déplacements et collisions :

Il existe des méthodes pour déplacer facilement ces rectangles :

La méthode move() :

persorect.move(dx,dy)

Cette méthode s'applique à un objet rect (ici persorect) et contient 2 arguments correspondant au nombre de pixels dont on veut déplacer l'objet Rect()

Exemple :

persorect.move(3,0)

Déplacera le rectangle de 3 pixels vers la droite. Il **ne** faut **pas oublier de réassigner** cette nouvelle position à notre rectangle

persorect= persorect.move(3,0)

Ensuite à vous de gérer avec des évènements pour déplacer le personnage avec des flèches.

Attention tant qu'on déplace les rectangles à l'écran, il ne se passe **RIEN**.
Pour **afficher** les modification, il faudra « **bliter** » l'image sur la fenêtre à l'endroit où se trouve notre objet Rect()

fenetre.blit(perso,persorect)

*Coller sur la **fenêtre** l'image **perso**, à la position du rectangle **persorect***

*Pour **éviter les sorties d'écran**, on peut utiliser les **bords**(top, bottom, left, right) des rectangles comme référence.*

Première application :

A partir du fichier « ***base_ex3_a_completer.py*** », essayer de créer un programme plaçant initialement le personnage au milieu et permettant de le déplacer avec les flèches sans que celui-ci sorte de l'écran :(***correction en Annexe : « base_ex3_application1.py »***)

Deuxième application (facultatif en fonction du temps dont vous disposez) :

Programmer un déplacement de votre personnage à l'aide d'un vecteur vitesse, et faites-le rebondir sur les parois de la fenêtre.

Créer un vecteur vitesse à partir d'une liste nommée vitesse contenant 2 valeurs correspondants à la vitesse de déplacement suivant x et y (nombre de pixels suivants x ou y)
Exemple : **vitesse=[3,2]**

Utiliser les valeurs de la vitesse en x et en y pour déplacer votre objet Rect() correspondant à votre personnage.
Exemple : persorect=persorect.move(vitesse)

(***correction en Annexe : « base_ex3_application2.py »***)

Les collisions :

On reprend le programme de l'application 1 :

On charge l'image **mur** et on crée l'objet Rect() correspondant que l'on nommera **murrect.**
On positionne et on affiche le mur à un endroit quelconque sur la fenêtre.

```
mur=pygame.image.load("mur.png")
murrect=mur.get_rect()
murrect.x=300
murrect.y=200
```

La méthode colliderect() et copy():

Pour savoir si 2rectangles sont en collision, on utilise la méthode **colliderect()** qui renvoie **True** si les 2 rectangles se chevauchent

Ici en rouge l'objet Rect() correspondant à **murrect** et en bleu l'objet Rect() correspondant à **persorect**

Pour savoir s'il y a collision : on écrit l'instruction

```
If persorect.colliderect(murrect) :
    Faire les instructions…
```

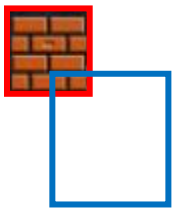
On aurait pu écrire également :

```
If murrect.colliderect(persorect) :
    Faire les instructions…
```

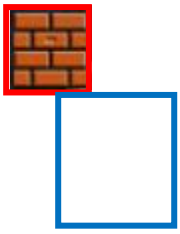
Ben oui mais je suis bien avancé avec tout çà ! qu'est-ce que j'en fais ?

Si je veux que mon personnage soit <u>bloqué</u> par le mur :
Si une détection de collision est faite, on va tout simplement demander de repositionner le personnage dans la position précédente. Ce qui signifie que l'on a besoin de connaitre cette ancienne position.

position lors de la collision



ancienne position avant la position



Pour connaitre à tout moment l'ancienne position, on va créer une copy de la position de notre personnage. à l'aide de la méthode copy(). Il suffit d'écrire :

```
oldpersorect=persorect.copy()
```

On a donc créé un nouveau rectangle **oldpersorect** qui est une copie du rectangle contenant le personnage (il contient donc la position et la taille du rectangle du personnage)

On aura donc à tout moment stocker l'ancienne position du rectangle correspondant à l'image du personnage.

<u>Il suffit d'écrire :</u>

```
if persorect.colliderect(murrect)==True:
    persorect=oldpersorect
    print"collision"
else:
    oldpersorect=persorect
    print"deplacement libre"
```

Si on détecte une collision, la position du personnage prend l'ancienne position

Si pas de collision :
on garde la nouvelle position persorect et on met à jour l'ancienne position

(***correction en Annexe :*** « base_ex3_application3_collision_v1.py »