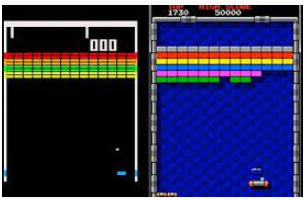
Projet Python - Le casse-briques

*Les casses-briques*

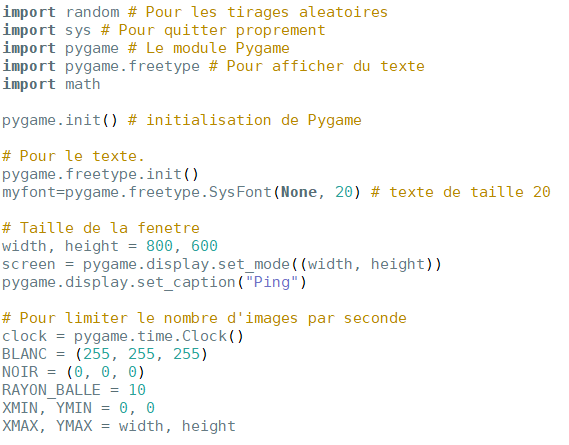
Faire une recherche historique sur ce jeu mythique et en faire un résumé que vous placerez dans un fichier que vous nommerez casse\_briques\_commentaires\_NOM\_PRENOM (fichier texte de votre choix lisible facilement).

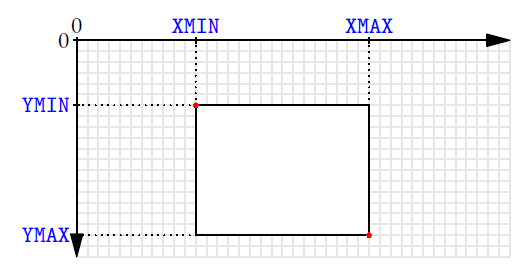


Le but de ce projet est de recréer un casse-brique en Python en utilisant la bibliothèque Pygame. La première partie consiste à faire une version très basique et la seconde permettra de l’enrichir afin d’obtenir un jeu plus intéressant. Vous devez utiliser la structure donnée, mais en aucun cas utiliser un autre programme avec d’autres noms de variables. Ce projet est à rendre au retour des vacances de février (jeudi 23/02/2023) de manière individuelle et personnalisée.

*La balle*

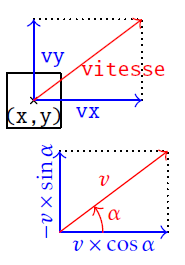
Nous allons commencer par mettre en place les bases de l’affichage et les rebonds de la balle sur les bords. Dans un fichier casse\_briques\_NOM\_PRENOM.py, recopier les lignes suivantes :



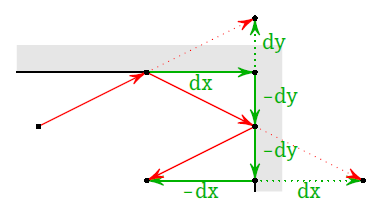


Les premières lignes servent à inporter et à configurer les différentes bibliothèques, donc Pygame. Les dernières servent à définir les constantes. Les coordonnées utilisées par Pygame sont exprimées en pixels. L’origine du repère se trouve en haut à gauche de l’écran, l’axe des abscisses va vers la droite et celui des ordonnées vers le bas. La balle se déplacera dans le rectangle dont le coin haut gauche est (XMIN, YMIN) et le coin bas droite est (XMAX, YMAX).

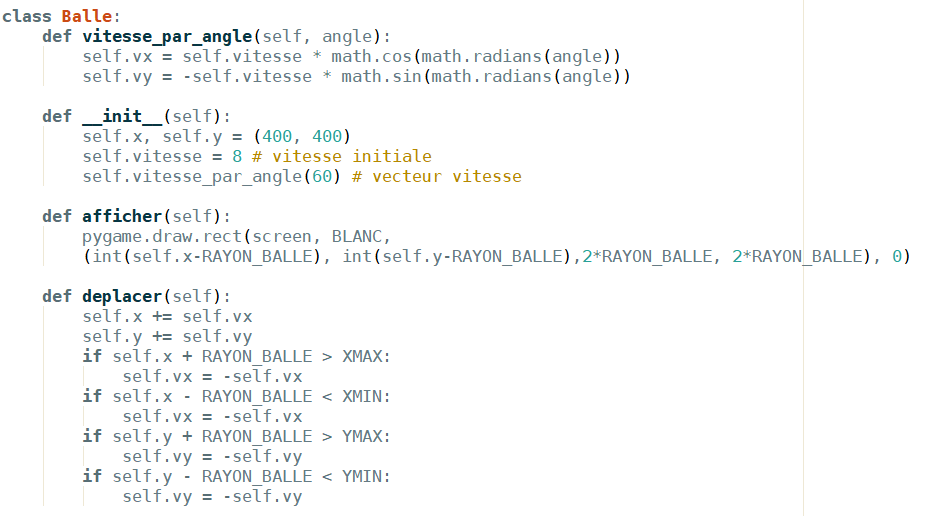
Pour l’instant nous allons utiliser l’ensemble de la surface disponible à l’écran.

Une balle est définie par 5 attributs : x et y pour sa position, vx et vy pour le vecteur de déplacement, dont la norme est vitesse. À chaque étape, la balle avance de vx sur les abscisses et de vy sur les ordonnées. Afin de calculer le vecteur initial, il faut faire un peu de trigonométrie. Les fonctions cos et sin de math prennent des angles exprimées en radians, c’est pourquoi il faut les convertir. Les ordonnées étant orientées vers le bas, il faut prendre l’opposé de l’ordonnée du vecteur pour avoir les angles dans le sens trigonométrique classique.

Pour l’affichage, nous nous contentons d’un rectangle blanc centré en (x, y) et dont le côté mesure 2 \_ RAYON\_BALLE.

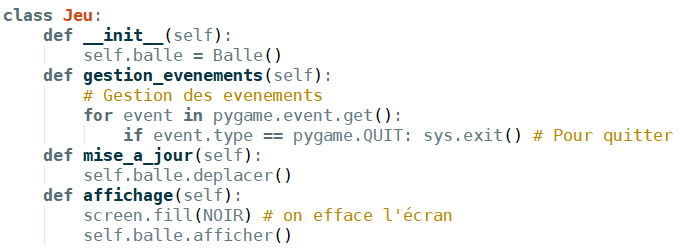
Lorsqu’on met à jour la position de la balle, on vérifie si elle rebondit contre un des murs de la zone. Pour cela, il faut juste vérifier si le bord droit de la balle dépasse XMAX, le bord gauche est inférieur à XMIN. . . Selon les cas, on prend l’opposé de vx ou de vy.

La balle est définie par la classe suivante :

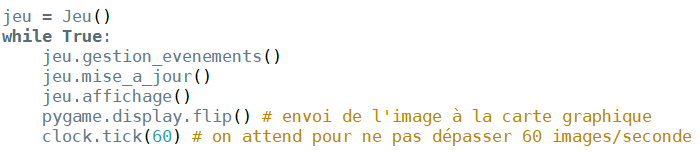


Afin de voir évoluer la balle, nous rajoutons une classe Jeu qui s’occupera de la boucle principale du jeu :

Celle classe servira à gérer les actions du joueur, les interactions entre les éléments et à afficher la nouvelle image. Pour l’instant la seule action que peut faire le joueur est de quitter en cliquant sur la croix de la fenêtre. Le seul élément à l’écran est la balle qui rebondit sur les bords de la fenêtre.



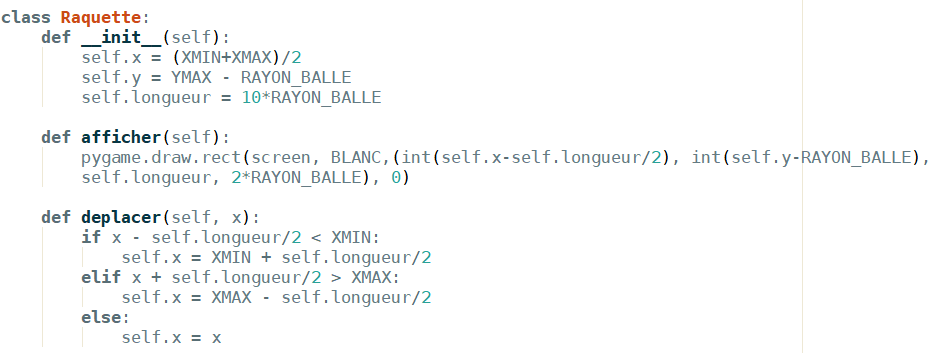
Pour finir le programme, il faut rajouter les lignes suivantes :



Exercice 1 : Recopier les différentes parties dans votre fichier et vérifier que la balle rebondit bien contre les bords.

*La raquette*

Pour ce prototype, la raquette sera contrôlée par l’abscisse de la souris. Plus tard, vous pourrez rajouter des contrôles au clavier. Commençons par l’afficher et gérer ses déplacements :



Les coordonnées correspondent au centre de la raquette. La hauteur de la raquette n’est pas indiquée puisque c’est le double du rayon de la balle. Afin de la déplacer, nous vérifions qu’elle ne sort pas de la zone prévue.

Il faut également mettre à jour la classe Jeu. Voici les changements :



Exercice 2 : Mettre à jour le fichier et tester le déplacement de la raquette.

*Rebond sur la raquette*

Il faut maintenant gérer les collisions entre la balle et la raquette. Contrairement aux briques que nous rajouterons plus tard, toute collision entre les deux renverra automatiquement la balle vers le haut. Il suffit donc de savoir si la balle et la raquette se superposent. Pour cela, nous pourrions utiliser les masques fournis par Python qui permettent de tester le chevauchement de deux images, mais nous allons utiliser une technique plus simple qui sera plus pratique avec les briques. La balle et la raquette se touchent si l’écart entre le centre de la raquette et le centre de la balle est inférieur à *r* + *l=*2 pour les abscisses et 2*r* pour les ordonnées, où *r* est le rayon de la balle et *l* la longueur de la raquette. Il faut rajouter ce test avant le test du rebond sur les murs, pour éviter des problèmes plus tard.

Il faut modifier ou rajouter les méthodes suivantes :

class Balle:

...

def deplacer(self, raquette):

self.x += self.vx

self.y += self.vy

if raquette.collision\_balle(self):

self.vy = -self.vy

if self.x + RAYON\_BALLE > XMAX:

self.vx = -self.vx

...

class Raquette:

...

def collision\_balle(self, balle):

vertical = abs(self.y - balle.y) < 2\*RAYON\_BALLE

horizontal = abs(self.x - balle.x) < self.longueur/2 + RAYON\_BALLE

return vertical and horizontal

class Jeu:

...

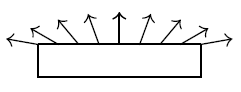
def mise\_a\_jour(self):

x, y = pygame.mouse.get\_pos()

self.balle.deplacer(self.raquette)

self.raquette.deplacer(x)

Exercice 3 : Modifier le programme et vérifier que la balle rebondit bien sur la raquette.

L’angle de la balle est toujours le même, ce qui peut rendre le jeu un peu monotone. Pour donner plus de possibilité au joueur, la direction dans laquelle repart la balle dépend de l’endroit où elle touche la raquette. Elle repart vers la droite si elle la touche à droite et vers la gauche sinon. Plus la balle se trouve proche des bords et plus la direction se rapprochera de l’horizontal. Pour cela, on rajoute la fonction

class Balle:

...

def rebond\_raquette(self, raquette):

diff = raquette.x - self.x

longueur\_totale = raquette.longueur/2 + RAYON\_BALLE

angle = 90 + 80 \* diff/longueur\_totale

self.vitesse\_par\_angle(angle)

def deplacer(self, raquette):

self.x += self.vx

self.y += self.vy

if raquette.collision\_balle(self) and self.vy > 0:

self.rebond\_raquette(raquette)

...

Les angles possibles vont alors de 10 à 170. Nous rajoutons un test pour la collision avec la raquette. Il faut que la balle descende, sinon, nous ne testons pas la collision avec la raquette pour éviter qu’elle ne change d’angle plusieurs fois.

Exercice 4 : Vérifier cette nouvelle version.

*Game Over*

Actuellement, le sol est considéré comme n’importe quel autre mur. Nous allons modifier le comportement pour que la balle revienne sur la raquette et qu’elle soit lancée en appuyant sur le bouton gauche de la souris. Pour cela, on rajoute un attribut à la balle indiquant si elle est collée à la raquette ou pas.

class Balle:

...

def \_\_init\_\_(self):

self.x, self.y = (400, 400)

self.vitesse = 8 # vitesse initiale

self.vitesse\_par\_angle(60) # vecteur vitesse

self.sur\_raquette = True

def deplacer(self, raquette):

if self.sur\_raquette:

self.y = raquette.y - 2\*RAYON\_BALLE

self.x = raquette.x

else:

self.x += self.vx

self.y += self.vy

if raquette.collision\_balle(self) and self.vy > 0:

self.rebond\_raquette(raquette)

if self.x + RAYON\_BALLE > XMAX:

self.vx = -self.vx

if self.x - RAYON\_BALLE < XMIN:

self.vx = -self.vx

if self.y + RAYON\_BALLE > YMAX:

self.sur\_raquette = True

if self.y - RAYON\_BALLE < YMIN:

self.vy = -self.vy

class Jeu:

...

def gestion\_evenements(self):

# Gestion des evenements

for event in pygame.event.get():

if event.type == pygame.QUIT: sys.exit() # Pour quitter

elif event.type == pygame.MOUSEBUTTONDOWN: # On vient de cliquer

if event.button == 1: # Bouton gauche

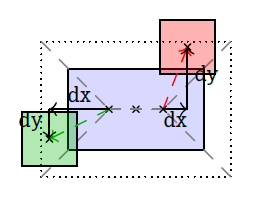
if self.balle.sur\_raquette:

self.balle.sur\_raquette = False

self.balle.vitesse\_par\_angle(60)

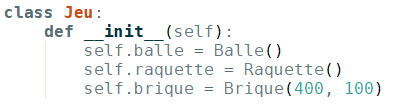
Exercice 5 : Faites les modifications et tester votre programme.

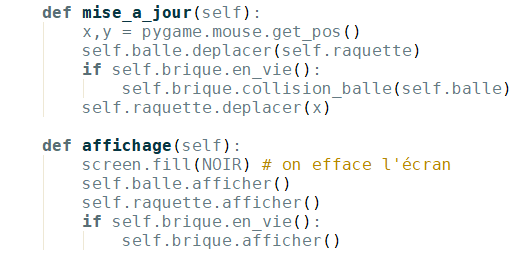
*Les briques*

Il faut maintenant ajouter les briques. Contrairement à la raquette, il faut regarder de quel côté tape la balle pour savoir s’il faut faire un rebond horizontal ou un rebond vertical. Pour cela, nous allons définir 4 zones autour de la brique afin de déterminer où se trouve la balle au niveau du rebond. On commence par regarder si la balle se trouve à droite ou à gauche de la brique. On se place ensuite sur le point du côté correspondant de la brique se trouvant à égale distance du haut, du bas et du côté de la brique. On regarde l’écart qu’il y a entre ce point et le centre de la balle. Selon la coordonnée la plus grande de ce vecteur, en valeur absolue, on peut déterminer si la balle se trouve sur les côtés ou en haut ou en bas de la brique.



Puis on modifie la classe Jeu :





Exercice 6 : Rajouter la classe Brique et tester la nouvelle version du jeu.

Il se peut que les rebonds ne soient pas ceux attendus, surtout lorsque la balle arrive presque perpendiculairement à un des angles. Il faudrait utiliser une méthode plus précise pour déterminer le côté de la brique qui est tapé en premier.

*Les premiers ajouts*

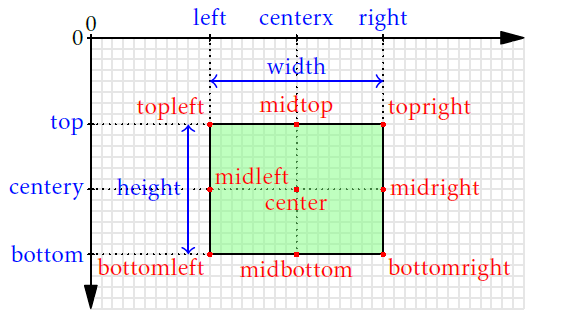
Le jeu n’est pas encore vraiment un vrai jeu. Il manque de nombreux éléments. Voici les premiers à ajouter pour avoir la version de base du jeu. L’objectif est de se concentrer sur les aspects algorithmiques sans perdre du temps sur les graphismes. Vous pouvez avoir besoin d’afficher du texte. Pour cela, vous pouvez utiliser le code suivant :

texte, rect = myfont.render("Mon message", COULEUR, size=TAILLE)

rect.midleft = COORD\_POINT

screen.blit(texte, rect)

Vous pouvez changer le texte, la couleur, la taille (20 par défaut) et les coordonnées du point à partir duquel vous écrivez le texte. Avec ces instructions, le texte est aligné à gauche. Vous pouvez l’aligner autrement en choisissant un autre point d’encrage pour le rectangle contenant le texte. Vous pouvez utiliser le schéma suivant pour connaître les différents points que vous pouvez utiliser :



Si vous avez besoin d’autres informations, vous pouvez tout retrouver dans la documentation de Pygame : https://www.pygame.org/docs/

Voici ce que vous devriez ajouter en priorité :

• Ajouter plus de briques : Au lieu d’associer une seule brique à la classe Jeu, il faut lui associer une liste de briques permettant d’avoir un niveau complet. Il faudra peut-être voir s’il est nécessaire de gérer les rebonds de la balle lorsqu’elle touche plusieurs briques en même temps.

• Rajouter des vies : Le jeu n’est pas très intéressant si on ne peut pas perdre. Rajouter un attibut vies à Jeu et modifier le jeu pour qu’il s’arrête lorsque les vies tombent à 0. Vous pouvez alors afficher le message “Game Over”. Un clic de souris permettrait alors de réinitialiser le jeu. Il pourrait être judicieux de déplacer l’initiation de la classe Jeu dans une autre méthode qui serait appelée par \_\_init\_\_ et lorsqu’il faut recommencer une partie.

• Ajouter le score : Il sera probablement nécessaire de modifier les valeurs XMIN et XMAX afin de laisser de la place à l’écran pour afficher le score. Il faudra alors dessiner des bords pour la zone de jeu afin de visualiser les murs.

• Ajouter des niveaux : Lorsque tous les briques sont détruits, le jeu passe au niveau suivant en modifiant la liste des briques du jeu.

Avec cela, vous avez déjà de quoi vous amuser.

*Pour aller plus loin*

Une fois que la base du jeu est fonctionnelle, vous pouvez commencer à rajouter des éléments de gameplay. Voici quelques suggestions, mais vous pouvez aussi rajouter vos propres idées :

• Des briques indestructibles ou qui nécessitent plusieurs coups pour être détruits.

• La balle peut accélérer au fur et à mesure du jeu.

• Des bonus qui tombent des briques détruits et qui permettent d’allonger la raquette, ralentir la balle, tirer sur les briques, traverser les briques sans rebondir. . .

• Faire gagner des vies au bout d’un certain nombre de points.

• Rajouter des continus.

• Mettre des couleurs différentes pour les briques.

• Contrôler la raquette au clavier.

• Faire des briques qui bougent.

• Améliorer les collisions entre la balle et les briques pour avoir quelque chose de plus précis.

• Rajouter des ombres sous les différents éléments pour donner plus de profondeur aux graphismes.

• Rajouter des sons.

• Utiliser des images pour les différents éléments. Il faudra peut-être revoir les conditions des collisions.

• Rajouter un mode 2 joueurs, à tour de rôle.

Il y a bien entendu plein d’autres idées possibles.