Chapitre 2

Pygame : Construire des jeux en Python

Package **pygame**

Pygame est une libraire qui permet de réaliser des jeux vidéo en Python. Elle s'importe comme n'importe quelle librairie.

Pygame doit ensuite être initialisé.

```
import pygame

pygame.init()
```

Création d'une fenêtre

Créer une fenêtre se fait de la manière suivante :

```
# Création d'une fenêtre
fenêtre = pygame.display.set_mode((640, 480))

# Ceci permet de donner un titre à la fenêtre
pygame.display.set_caption("Hello World")
```

Le paramètre **(640, 480)** de **set_mode** définit la taille en pixels de la fenêtre.

Boucle principale du jeu

Pygame est conçu pour avoir une boucle infinie (**`while True`**) qui vérifie à chaque tour si de nouveaux événements se sont produits, et qui agit en fonction des événements. C'est dans cette boucle que le contenu de la fenêtre va être créé.

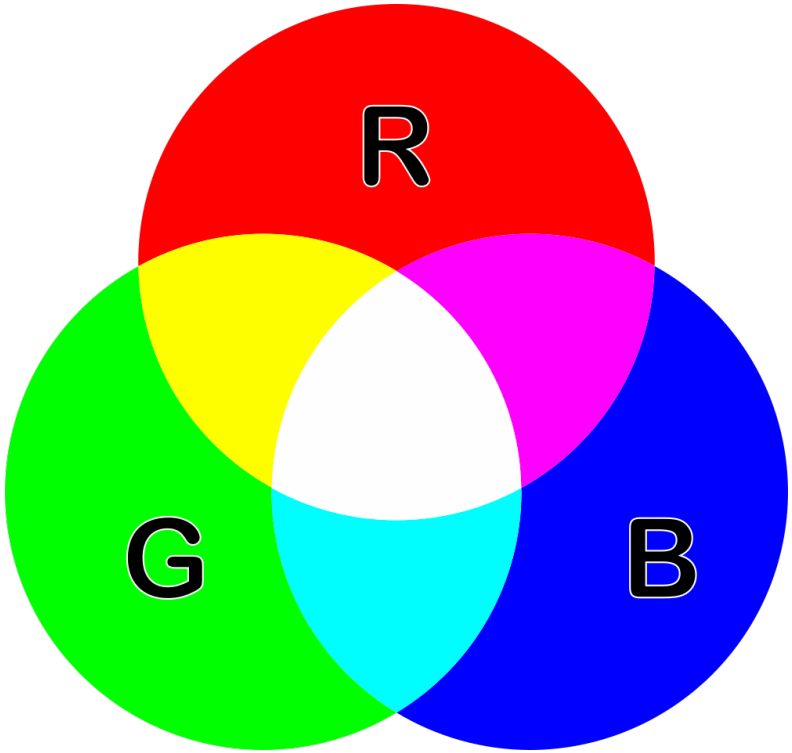
```
import pygame
import sys
...

# Boucle infinie
while True:
    # Parcourt de tous les événements
    for event in pygame.event.get():
        # Vérifie si l'événement est "quitter"
        if event.type == pygame.QUIT:
            # Met fin au programme
            pygame.quit()
            sys.exit()
```

Exercice 1

- Crée ton premier programme **pygame**
- Ouvre une fenêtre de la taille 800×600 pixels.
- Donne ton nom à la fenêtre.
- Lorsqu'on ferme la fenêtre, le programme doit s'arrêter.

Les couleurs en informatique

Un écran d'ordinateur est composé de pixels. Chaque pixel est composé de 3 lampes : une rouge (R), une verte (G) et une bleue (B). En fonction de la luminosité de chaque lampe, une couleur est perçue par notre œil.



Les valeurs de chaque lampe sont comprises entre 0 et 255.

Par exemple, ($R = 255$, $G = 127$, $B = 0$) donne du orange.

`https://rgbcolorpicker.com/`

Dessiner dans la fenêtre

La fenêtre est une surface dans laquelle on peut dessiner des éléments.

```
# Définir des couleurs
blanc = (255, 255, 255) # R=255 ; G=255 ; B=255
orange = (255, 127, 0)  # R=255 ; G=127 ; B=0

# Dessiner des rectangles
# pygame.draw.rect(surface, couleur, (x, y, larg., haut.))
pygame.draw.rect(fenêtre, blanc, (10, 20, 30, 40))
pygame.draw.rect(fenêtre, orange, (200, 300, 50, 50))

# À ne pas oublier pour appliquer les changement
pygame.display.update()
```

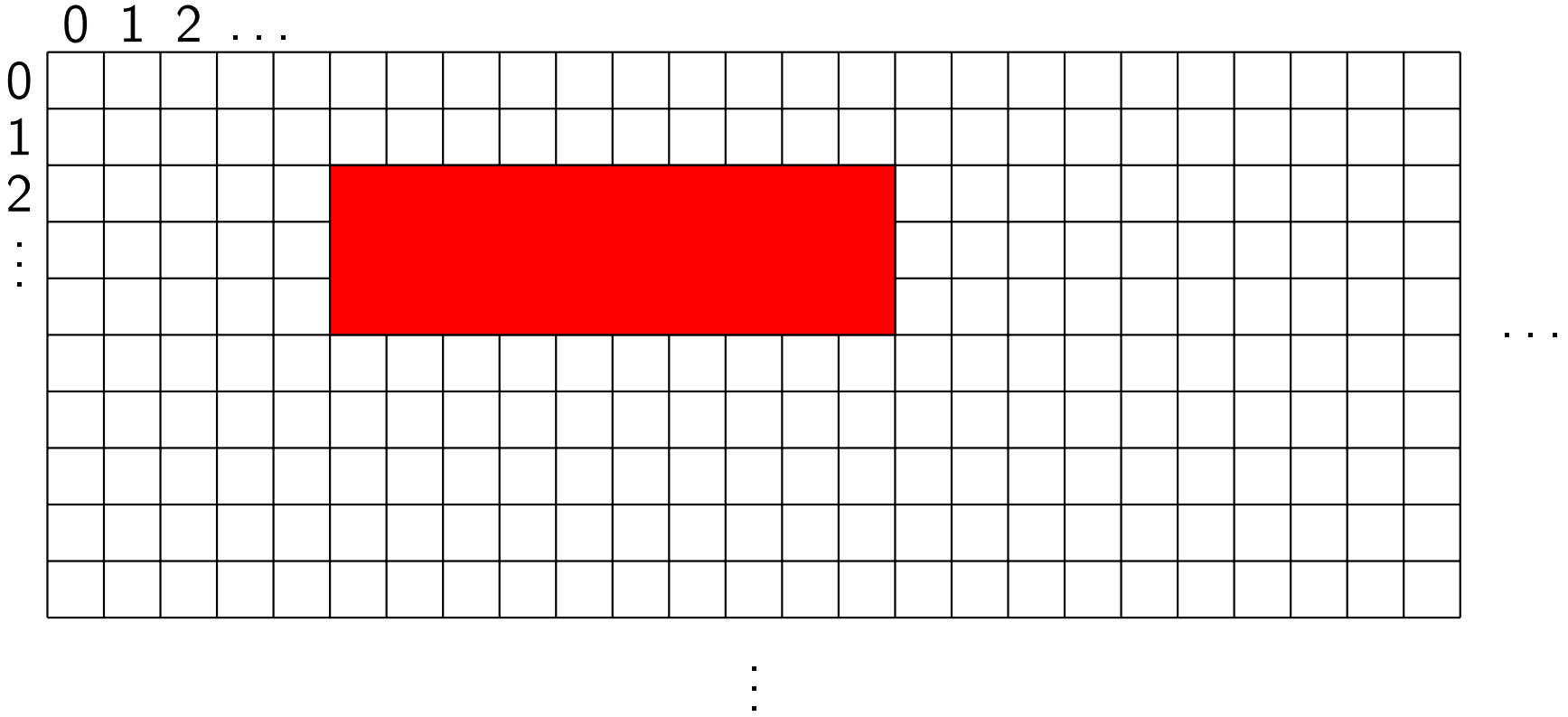
Exercice 2

- Dans un programme **`pygame`**, trace un rectangle rouge de 200 pixels de large et de 100 pixels de haut, à la position (100, 10)
- Trace trois autres rectangles de trois autres couleurs au choix.

- Dans quel coin de la fenêtre se trouve la position $(0, 0)$?

Les coordonnées de la fenêtre

Voici une représentation agrandie des pixels d'une fenêtre **pygame** :




```
# Tracer ce rectange rouge, dont les coordonnées du
# coin supérieur gauche sont (5, 2) et les dimensions
# sont (10, 3)
pygame.draw.rect(fenêtre, rouge, (5, 2, 10, 3))
```

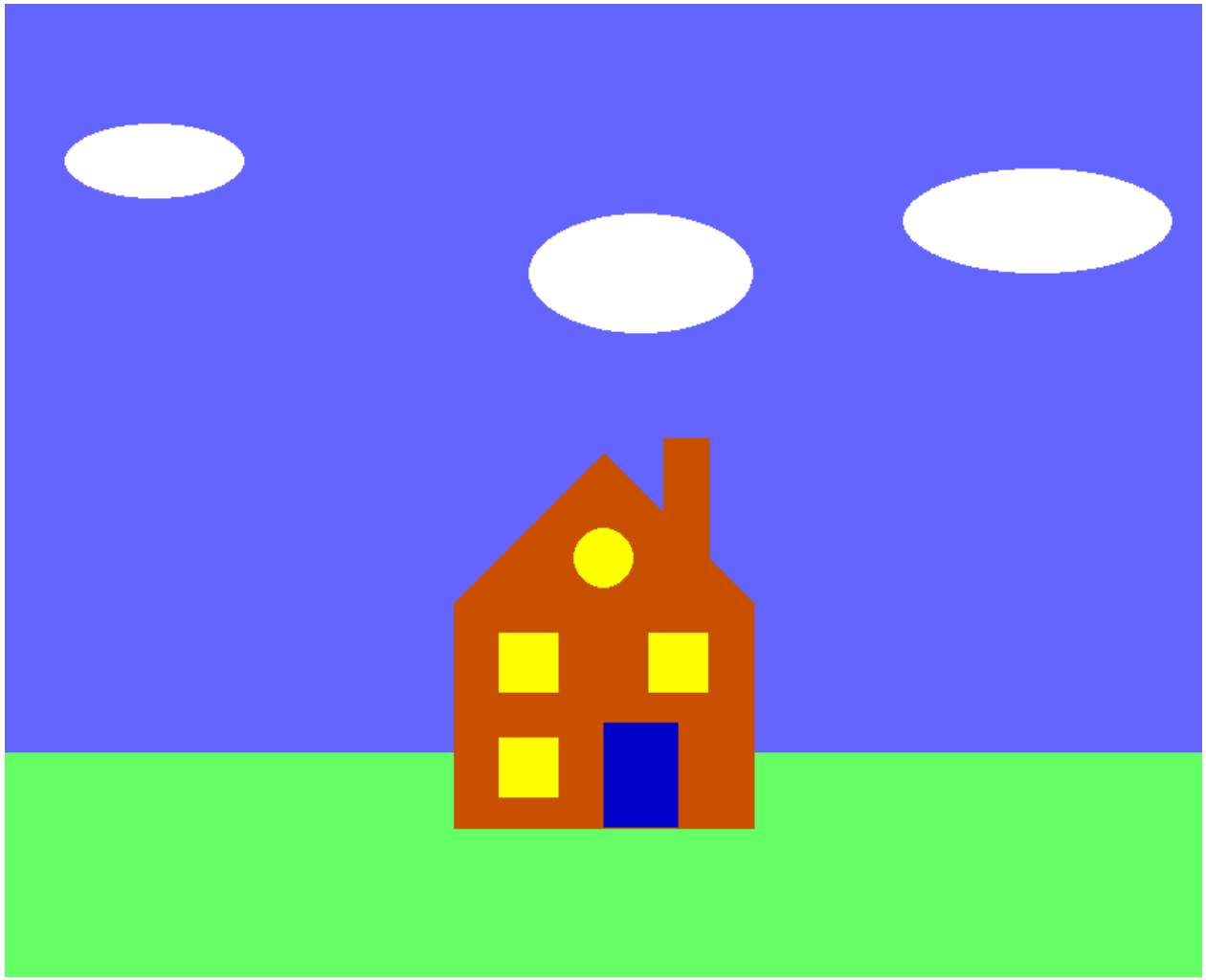
Formes

Il est possible de tracer d'autres formes qu'un rectangle :

```
# Cercle de centre (100, 150) et de rayon 30 :
pygame.draw.circle(fenêtre, blanc, (100, 150), 30)

# Ellipse comprise dans le rectangle (30, 30, 100, 50)
pygame.draw.ellipse(fenêtre, jaune, (30, 30, 100, 50))

# Ligne simple d'un point à un autre
pygame.draw.line(fenêtre, bleu, (20, 150), (300, 100))

# Polygone passant par les points de la liste
pygame.draw.polygon(fenêtre, vert,
          [(80, 60), (120, 85), (110, 120), (60, 100)])
```

Exercice 3

Écris un programme **pygame** qui donne le résultat suivant :



Intégrer des images

On peut charger une image depuis le disque

```
# Crée une surface contenant l'image chargée depuis
# un fichier
personnage = pygame.image.load('petit_chien.png')
```

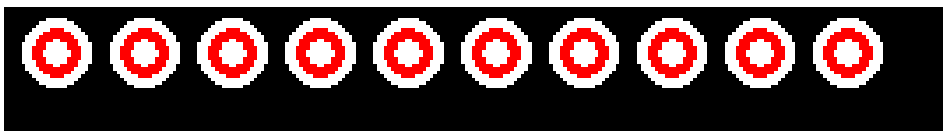
puis l'ajouter à l'écran

```
# Affiche le personnage à la position donnée (coin
# supérieur gauche de l'image)
fenêtre.blit(personnage, (100, 200))
```

Intégrer des images

Notez qu'on peut aussi créer sa propre image (surface), et ensuite la réutiliser.
`convert_alpha()` permet d'avoir un fond transparent.

```
# Génération de l'image, une seule fois au début
# du programme
balle = pygame.Surface((21, 21)).convert_alpha()
balle.fill((0,0,0,0)) # Fond transparent
pygame.draw.circle(balle, blanc, (10, 10), 10)
pygame.draw.circle(balle, rouge, (10, 10), 7)
pygame.draw.circle(balle, blanc, (10, 10), 4)

# Utilisation de l'image
for x in range(10):
    fenêtre.blit(balle, (5 + x*25, 5))
```



Exercice 4

- Télécharge au moins trois images depuis internet
- Crée une petite scène avec tes images à l'aide de **`pygame`**

Animation

Lorsqu'on crée un jeu vidéo, des éléments du jeu sont en mouvement. Créons une petite animation de notre personnage.

```
clock = pygame.time.Clock() # Horloge du jeu
posX = 0 # position du personnage
vitesse = 100 # pixels par seconde
while True:
    # Durée en secondes
    durée = clock.tick(30) / 1000.0
    # Calcul du déplacement
    déplacement = vitesse * durée
    # Déplacement
    posX += déplacement
    # Affichage
    fenêtre.fill((0,0,0)) # Écran noir
    fenêtre.blit(perso, (posX, 30)) # Personnage
    pygame.display.update()
```

`clock.tick(30)` permet de connaitre le temps (en millisecondes) passé depuis la dernière itération, et d'avancer notre personnage en conséquence selon sa vitesse. Le paramètre **`30`** permet de forcer l'animation à maximum 30 images par seconde.

Exercice 5

- Crée une petite animation d'un personnage qui avance horizontalement sur l'écran à vitesse constante.
- Lorsque le personnage atteint l'extrémité de l'écran, il doit repartir dans l'autre sens.

- Pour aller plus loin : donne à la vitesse de ton personnage une composante verticale et une composante horizontale. Le personnage rebondira sur les quatre bords de la fenêtre.

Le clavier

Dans un jeu vidéo, la joueuse ou le joueur interagit avec le jeu de différentes manières (manette, souris, clavier, joystick, voix, etc.). Nous allons nous attarder sur le clavier et la souris, puisque c'est le matériel que nous avons à disposition.

Pour intercepter un événement de clavier, nous pouvons utiliser les événements dans notre boucle principale :

```
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            ...
        elif event.type == KEYUP:
            # Une touche a été relâchée
            # Récupérons le symbole correspondant
            symbole = event.unicode
            if symbole == ' ': # Le symbol est un espace
                stop = True # Le jeu ou le personnage s'arrête
```

Événements du clavier

Déplaçons notre personnage (dans une grille de 20×20) lorsque les flèches du clavier sont utilisées.

```
from pygame.locals import * # Constantes (KEYDOWN, etc.)

posX, posY = 0, 0 # Position initiale
...
elif event.type == KEYDOWN:
    if event.key == K_UP: # Flèche haut
        if posY > 0:
            posY -= 1
    if event.key == K_DOWN: # Flèche bas
        if posY < 19:
            posY += 1
    if event.key == K_LEFT: # Flèche gauche
        if posX > 0:
            posX -= 1
    if event.key == K_RIGHT: # Flèche droite
        if posX < 19:
            posX += 1
...
# Affichage
fenêtre.blit(perso, (posX * 20, posY * 20))
```

Exercice 6

- Crée un programme **pygame** dans lequel un personnage est contrôlé par le clavier.
- Les flèches du clavier le font avancer, reculer, monter et descendre d'un nombre fixé de pixels.
- La touche 'espace' le fait se téléporter à un endroit aléatoire (utilise la libraire **random**).

- Pour aller plus loin, fais en sorte que lorsque le personnage quitte un côté de l'écran, il réapparaisse en face.

État du clavier

Pour savoir si une touche du clavier reste enfoncée, on peut garder une trace de son état grâce aux événements **KEYDOWN** et **KEYUP**.

Par exemple, essayons de savoir si la touche **X** est enfoncée :

```
touche_x_enfoncée = False # Initialement, la touche n'est pas
                          # enfoncée
...
while True:
    for event in pygame.event.get():
        if event.type == KEYDOWN: # Touche pressée
            if event.key == K_x: # Touche 'X'
                # La touche est maintenant enfoncée
                touche_x_enfoncée = True
        if event.type == KEYUP: # Touche relâchée
            if event.key == K_x: # Touche 'X'
                # La touche n'est maintenant plus enfoncée
                touche_x_enfoncée = False
    ...
    if touche_x_enfoncée:
        # La touche 'X' est actuellement enfoncée
    else:
        # La touche 'X' n'est pas enfoncée
```

Exercice 7

- Écris un programme **`pygame`** dans lequel le personnage peut être déplacé avec les flèches du clavier.
- Le personnage doit se déplacer tant que les touches sont enfoncées, et s'arrêter quand les touches sont relâchées.
- Il doit être possible de déplacer le personnage en diagonale si deux flèches sont maintenues enfoncées en même temps.

La souris

Deux interactions sont possibles avec la souris : le déplacement et les clics. Ces deux interactions sont gérées par des événements, de type **MOUSEMOTION**, **MOUSEBUTTONDOWN** et **MOUSEBUTTONUP**.

```
while True:
    for event in pygame.event.get():
        if event.type == MOUSEMOTION: # Souris déplacée
            print("La souris a été déplacée à la positon", event.pos)
        elif event.type == MOUSEBUTTONDOWN: # Bouton souris enfoncé
            print("Le bouton", event.button,
                  "a été enfoncé à la position", event.pos)
        elif event.type == MOUSEBUTTONUP: # Bouton souris relâché
            print("Le bouton", event.button,
                  "a été relâché à la position", event.pos)
```

Exercice 8

- Crée un programme **`pygame`** qui écoute les événements de la souris
- À chaque fois que la souris est déplacée, un petit point rouge est affiché à la position du pointeur de la souris
- À chaque fois que le bouton gauche est cliqué, un gros point bleu est affiché à la position du clic.

- Améliorations possibles pour aller plus loin :
  - Au lieu d'afficher un point, affiche une ligne depuis la dernière position de la souris, ainsi le chemin suivi par le pointeur de la souris sera tracé
  - Un clic n'est valable que si le bouton est enfoncé et relâché à la même position
  - D'autres éléments sont affichés pour le clic droit et le clic milieu.
  - Pour aller encore plus loin : des ronds grossissants apparaissent après chaque clic (comme à la surface de l'eau). Attention, dans ce cas il te faudra redessiner toute la fenêtre à chaque fois, et donc garder en mémoire tout le éléments à tracer (ou mieux : utiliser une surface intermédiaire pour tracer le chemin de la souris, et afficher les ronds par-dessus celle-ci)

Afficher du texte à l'écran

Pour afficher du texte dans le jeu, il ne suffit plus d'utiliser **print**, car le texte doit être intégré dans l'image. On utilise plutôt la technique suivante :

1. On écrit le texte dans une surface
2. On affiche la surface à l'écran

```
# Police de caractère
police = pygame.font.SysFont('arial', 16)
# Surface contenant le texte
surfaceTexte = police.render('Score : 12', True, blanc)
# Ajout du texte dans la fenêtre
fenêtre.blit(surfaceTexte, (10, 10))
```

Exercice 9

Écris un programme **pygame** dans lequel, à chaque fois que le joueur clique dans la fenêtre, une lettre au hasard apparait à l'endroit du clic.

Note : la fonction **chr** *permet de convertir un nombre en lettre. Ainsi,* **chr(65)** *correspond à* **A**, **chr(66)** *correspond à* **B**, *etc.*

Un premier jeu

Nous allons à présent créer ensemble un premier jeu. Dans ce jeu, des briques tomberont sur l'écran, et le joueur devra les attraper à l'aide de la souris.

Commençons par importer les packages nécessaires et définissons quelques constantes :

```
import pygame
import sys
import random
from pygame.locals import *

# Couleurs
BLANC = (255, 255, 255)
ROUGE = (255, 0, 0)
BLEU = (0, 0, 255)
NOIR = (0, 0, 0)
```

Un premier jeu : initialisation

Définissons à présent les paramètres constants du jeu :

```
# Paramètre du jeu
LARGEUR_FENÊTRE = 1000
HAUTEUR_FENÊTRE = 800
LARGEUR_BRIQUE = 100
HAUTEUR_BRIQUE = 40
VITESSE_BRIQUE = 100 # Pixels par seconde
PROBABILITÉ_NOUVELLE_BRIQUE = 0.05
```

Comme toujours, il faut initialiser le jeu, la fenêtre et l'horloge :

```
# Initialisation de Pygame
pygame.init()

# Initialisation de la fenêtre
fenêtre = pygame.display.set_mode((LARGEUR_FENÊTRE, HAUTEUR_FENÊTRE))
pygame.display.set_caption("Jeu des briques qui tombent")

# Horloge du jeu
clock = pygame.time.Clock()
```

Un premier jeu : variables globales

La seule donnée qui doit perdurer pendant le jeu est la liste des briques en train de tomber. Nous allons stocker cela comme une liste dont chaque élément est une liste de longueur 2, contenant la position (x, y) de la brique.

```
# Liste des briques
briques = []
```

Si nous voulons afficher le score, il faut également garder cette information. Préparons aussi la police de caractère qui permettra de l'afficher.

```
# Score
score = 0
POLICE_SCORE = pygame.font.SysFont('arial', 20)
```

Un premier jeu : boucle principale

Nous pouvons maintenant commencer à écrire la boucle principale du jeu. Dans cette boucle, nous devons :

- Gérer l'événement de fermeture de la fenêtre
- Gérer les événements de clics de la souris
- Créer de nouvelles briques
- Faire avancer les briques

Nous allons utiliser des fonctions pour gérer la plupart de ces opérations. Cela rendra le code plus lisible, et nous permet d'écrire la structure du code avant de s'intéresser aux détails.

Un premier jeu : boucle principale

```
# Boucle principale du jeu
while True:
    # Gestion des événements
    for event in pygame.event.get():
        # Événement fin de jeu
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
        # Événement clic souris
        elif event.type == MOUSEBUTTONDOWN:
            # Récupérer la position du clic
            sourisX = event.pos[0]
            sourisY = event.pos[1]
            # Appel de la fonction qui va gérer ce clic
            clic_souris(sourisX, sourisY)
    # Crée une nouvelle brique avec une certaine probabilité
    if random.random() < PROBABILITÉ_NOUVELLE_BRIQUE:
        # Appel de la fonction qui crée une brique
        nouvelle_brique()
    # Calcul du temps écoulé depuis le dernière fois
    tempsSecondes = clock.tick(30) / 1000
    # Appel de la fonction qui fait avancer les briques
    avancer_briques(tempsSecondes)
    # Appel de la fonction qui affiche le jeu dans la fenêtre
    affichage()
```

Un premier jeu : fonction **nouvelle_brique**

Il nous reste à écrire les quatre fonctions du jeu. Commençons par la fonction qui crée une brique.

Une nouvelle brique est créée au sommet de l'écran. Plaçons-la à une position aléatoire. La position retenue sera le coins supérieur gauche, qui peut se trouver entre 0 et une largeur de brique avant la fin de la fenêtre, soit **LARGEUR_FENÊTRE - LARGEUR_BRIQUE**.

```
# Ajoute une nouvelle brique dans le jeu
def nouvelle_brique():
    posX = random.randint(0, LARGEUR_FENÊTRE - LARGEUR_BRIQUE)
    posY = 0
    # Ajout d'une brique, ses coordonnées sont stockées dans
    # un tableau
    briques.append([posX, posY])
```

Un premier jeu : fonction **avancer_briques**

Faisons à présent défiler les briques. À chaque itération, les brique descendent d'une longueur qui dépend du temps écoulé et de leur vitesse.

```
# Fait descendre toutes les briques en fonction du temps écoulé
def avancer_briques(temps):
    for brique in briques:
        brique[1] += VITESSE_BRIQUE * temps
```

Un premier jeu : fonction **affichage**

Occupons-nous à présent de l'affichage. Nous souhaitons afficher chaque brique, ainsi que le score.
La position d'une brique est stockée dans le tableau de deux valeurs la représentant, et ses dimensions sont fixées dans nos paramètres.

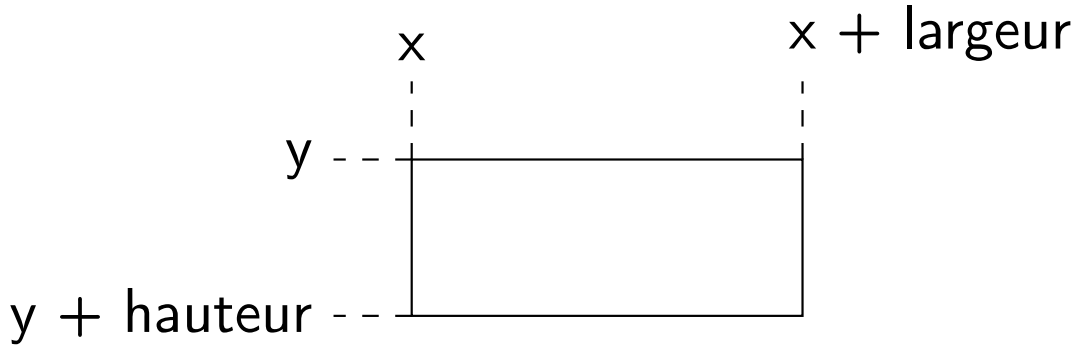
```
# Affiche tous les éléments du jeu
def affichage():
    # Fond noir
    fenêtre.fill(NOIR)
    # Briques
    for brique in briques:
        # Rectangle pour représenter la brique
        pygame.draw.rect(fenêtre, ROUGE, (brique[0], brique[1],
                                          LARGEUR_BRIQUE, HAUTEUR_BRIQUE))
    # Score
    scoreTexte = POLICE_SCORE.render(f"Score : {score}", True, BLANC)
    fenêtre.blit(scoreTexte, (10, 10))
    # Mise à jour de l'affichage
    pygame.display.update()
```

Un premier jeu : fonction **clic_souris**

Il nous reste à gérer les clics de la souris. Si un clic atteint une brique, celle-ci doit être supprimée et le score augmenté de 1.

Afin de ne pas perturber le parcours de liste lorsqu'une brique est supprimée, nous allons parcourir les briques de la dernière à la première. Utilisons pour ce faire une boucle **while**, et une variable **i** qui représentera l'indice (la position dans la liste) de chaque brique.

Une brique est atteinte si la position du clic se trouve dans son rectangle, c'est-à-dire que son abscisse est entre *x* et *x+largeur* et son ordonnée entre *y* et *y+hauteur* :



Un premier jeu : fonction **clic_souris**

```
# Supprime les briques atteintes par un clic de souris
# et fait évoluer le score
def clic_souris(sourisX, sourisY):
    # Variable globale que l'on souhait modifier
    global score
    # Dernier indice dans la liste des briques
    i = len(briques) - 1
    # Parcours des briques jusqu'à la première de la liste
    while i >= 0:
        # Brique à la position i dans la liste
        brique = briques[i]
        # Vérifie si la brique est atteinte par le clic
        if sourisX > brique[0] and \
           sourisX < brique[0] + LARGEUR_BRIQUE and \
           sourisY > brique[1] and \
           sourisY < brique[1] + HAUTEUR_BRIQUE:
               # La brique est atteinte, on la supprime
               del briques[i]
               # et on augmente le score de 1
               score += 1
        # On passe à la brique précédente
        i -= 1
```

Un premier jeu : gestion du *game over*

Il nous reste une dernière chose à faire pour ce jeu : gérer la fin de partie. Décidons qu'une partie se termine lorsqu'un brique tombe plus bas que la fenêtre. Il faut donc :

- Ajouter une variable globale **`gameOver = False`**.
- Dans la fonction **`avancer_briques`**, mettre cette variable à **`True`** si la position verticale de la brique (**`brique[1]`**) dépasse **`HAUTEUR_FENÊTRE`**.
- Dans la fonction **`affichage`**, il faut changer l'affichage en fonction de la valeur de **`gameOver`**, et afficher un message et le score lorsque la partie est terminée.
- Dans la boucle principale, la gestion des clics et des briques ne doit se faire que tant que **`gameOver == False`**.

Exercice 10

- Exécute et complète le jeu des briques qui tombent.

- Idées pour aller plus loin :
  - Remplace les briques rouges par des images de briques plus jolies, voire différents modèles ou couleurs de briques
  - La vitesse et la probabilité des nouvelles briques devraient augmenter avec le temps (ou avec les points)
  - Ajoute une animation visuelle lorsque le jeu est terminé
  - Toute autre idée que tu as pour améliorer le jeu

Projet

Il est temps à présent de créer ton propre projet **`pygame`**.

- Choisis ou invente les règles d'un jeu vidéo (vise la simplicité pour un premier projet)
- Code ton projet. Commence par la base, et ajoute progressivement des éléments du jeu, en testant au fur et à mesure ton avancée.
- N'hésite pas à demander de l'aide (privilégie l'aide d'un humain à celle d'une intelligence artificielle qui risque de te donner de mauvaises habitudes).

À propos de l'intelligence artificielle : l'IA peut être un outil extrêmement puissant en programmation. Mais, comme dans tout domaine, l'IA n'est qu'un assistant qui fait des erreurs, pas un professeur. N'utilise une IA que si tu es capable de discerner ses erreurs. **Bref, à ton stade d'apprentissage de la programmation, n'utilise pas encore d'IA.** Nous aurons l'occasion de revenir sur l'utilisation et le fonctionnement des IA plus tard dans ce cours.