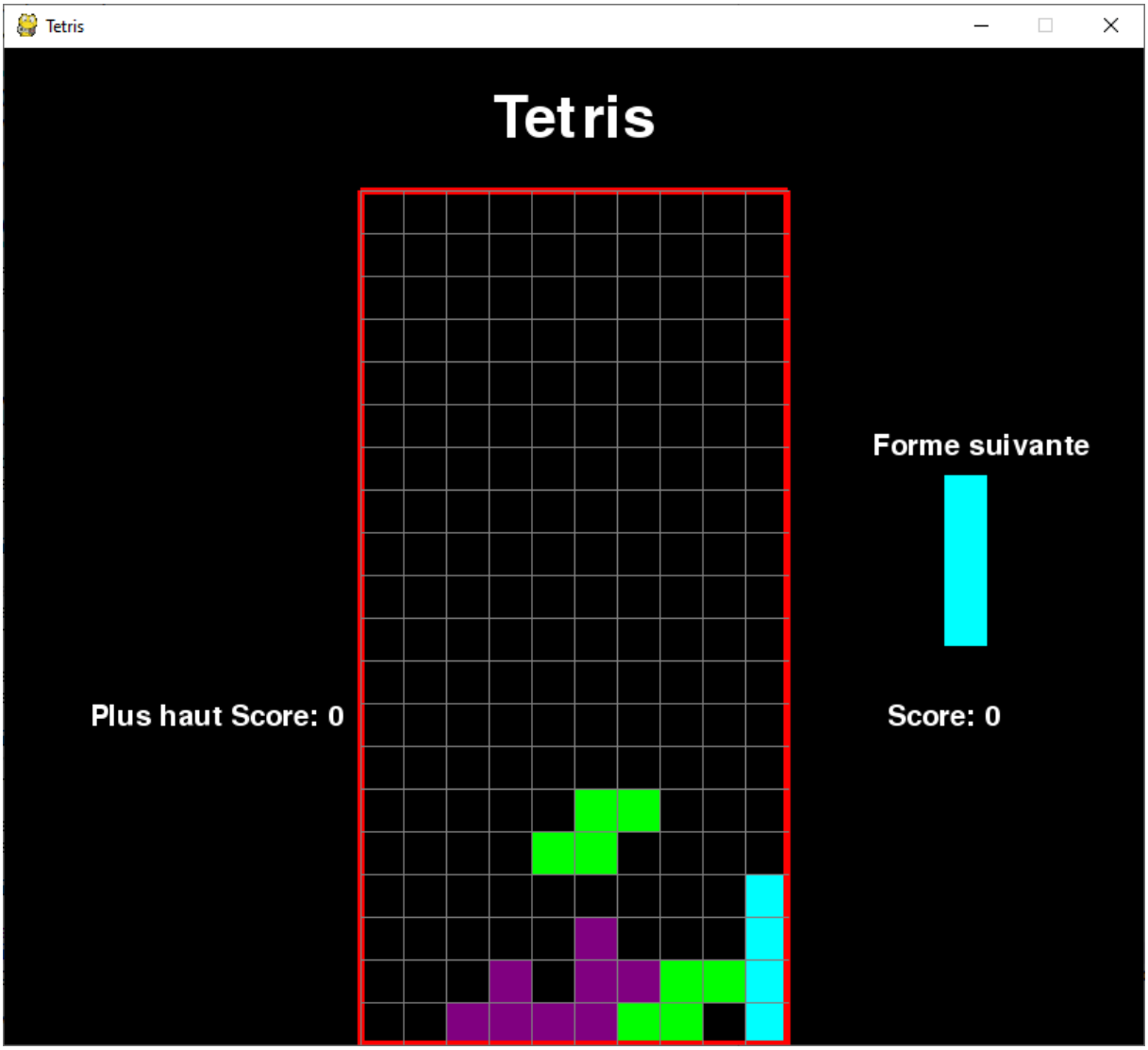
Tétris 3

Le jeu



Tester les emplacements occupés par une pièce lorsqu'elle tombe

Lorsque la pièce se déplace, tourne, nous devons nous assurer qu'elle ne vient pas occuper une place déjà prise par une autre pièce.

Nous allons créer une fonction espace_libre, qui prendra deux paramètres : la grille et la piece, et qui va tester si les emplacements de la grille, qui normalement seront occupés par la pièce, sont bien libres.

```
def espace_libre(piece, grille):
    #on recherche dans la grille les coordonnées (colonne,ligne)
    #des cases dont la couleur est noire
    position_possible=[]
    for i in range(nb_lignes):
        for j in range(nb_colonnes):
            if grille[i][j] == (0,0,0):
                t=(j,i)
                position_possible.append(t)
    #positions des cases colorées de la forme
    position_forme = convertir_forme(piece)
    #on regarde si au moins une position de la forme ne convient pas
    for pos in position_forme:
        if pos not in position_possible:
            #au démarrage de la forme, le numéro de ligne peut être négatif
            #il ne faut donc pas en tenir compte
            if pos[1] > -1:
                return False
    return True
```

Test:

Dans la boucle de jeu, lors de la prise en compte des évènements, nous testons si la piece peut occuper les positions dans la grille.

```
for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False
        #Tester si l'évènement CHUTE_PIECE a été déclenché.
        if event.type == CHUTE_PIECE:
            #faire descendre la pièce d'une ligne
```

```
            piece_enCours.y+=1
            if not espace_libre(piece_enCours, grille) :
                piece_enCours.y-=1
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                piece_enCours.x -= 1
                if not espace_libre(piece_enCours, grille) :
                    piece_enCours.x += 1
            if event.key == pygame.K_RIGHT:
                piece_enCours.x += 1
                if not espace_libre(piece_enCours,grille) :
                    piece_enCours.x -= 1
            if event.key == pygame.K_DOWN:
                piece_enCours.y += 1
                if not espace_libre(piece_enCours,grille) :
                    piece_enCours.y -= 1
            if event.key == pygame.K_UP:
                piece_enCours.rotation += 1
                if not espace_libre(piece_enCours,grille) :
                    piece_enCours.rotation -= 1
```

Nous pouvons remarquer que lorsque la pièce arrive au fond, elle ne peut plus descendre, mais elle peut encore se déplacer à droite ou à gauche.

Bloquer une pièce et envoyer la piece suivante

Lorsqu'une pièce arrive au fond, il faut maintenant bloquer sa position et envoyer une autre pièce.

Nous utiliserons une variable changer_piece qui sera à True lorsque lors de la descente on aboutit à des emplacements occupés.

```
    #Tester si l'évènement CHUTE_PIECE a été déclenché.
    if event.type == CHUTE_PIECE:
        #faire descendre la pièce d'une ligne
        piece_enCours.y+=1
        if not espace_libre(piece_enCours,grille) :
            piece_enCours.y-=1
            changer_piece = True
```

En fin de boucle nous ajoutons les cases de la pièce bloquée dans la liste des emplacements bloqués.

```
    #----------------------------------------------------------------------
    #Récupérer les positions des cases colorées de la pièce
    position_piece = convertir_forme(piece_enCours)
    #Ajouter la couleur de ces cases dans la grille à afficher
    for i in range(len(position_piece)):
        x, y = position_piece[i]
        if y > -1:
            grille[y][x] = piece_enCours.couleur
    #Changer de piece
    if changer_piece:
            #Mémoriser les cases occupées par la piece bloquée
            for pos in position_piece:
                p = (pos[0], pos[1])
                emplacement_bloque[p] = piece_enCours.couleur
            #choisr une autre pièce
            piece_enCours = choix_piece()
            changer_piece = False
```

#Mettre à jour la fenêtre

afficher_fenetre(ecran,grille)

pygame.display.update()

Effacer une ligne

Lorsqu'une ligne est pleine, la règle du jeu dit que cette ligne doit être effacée. Toutes les lignes au-dessus descendent d'une ligne.

La fonction effacer_ligne doit donc tester si une ligne est pleine.

Pour chaque ligne de la grille, nous devons tester s'il existe des cases dont la couleur est noire.

Si oui, nous devons mémoriser le numéro de la ligne et supprimer dans la liste des emplacements bloqués, toutes les cases de cette ligne.

Enfin, nous devons faire descendre d'un cran, les lignes situées au dessus de la ligne supprimée.

Voir Key Functions ici : https://docs.python.org/3/howto/sorting.html

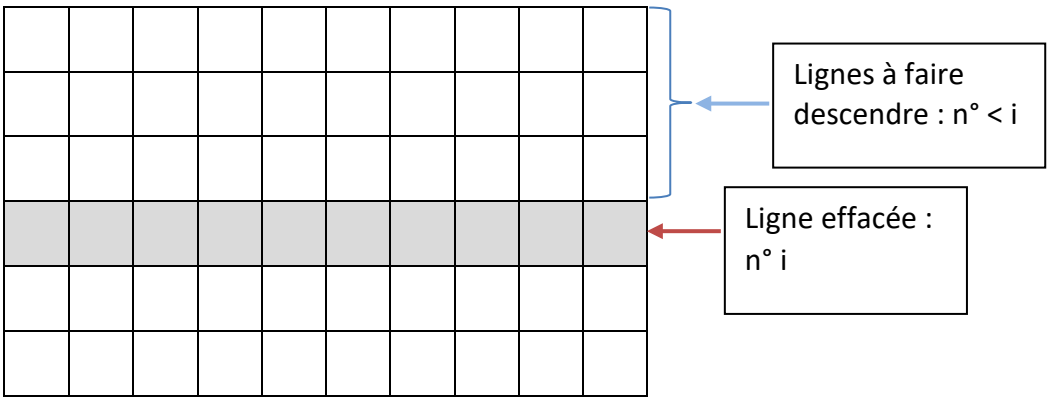
Voir https://docs.python.org/2.3/whatsnew/section-slices.html

Voir https://docs.python.org/fr/3/tutorial/datastructures.html

Processus d'effacement d'une ligne

Rappel : la variable « emplacement_bloque » est un dictionnaire dont les clés sont composées d'un tuple (n°colonne, n°ligne).

Connaissant le numéro de la ligne à effacer, il est facile avec la méhode del, d'effacer dans ce dictionnaire tous les éléments ayant dans leur clé, ce numéro de ligne.



Pour « faire descendre » une ligne, il suffit dans le dictionnaire de modifier le numéro de ligne, dans chaque clé dont le numéro de ligne actuel est inférieur à celui de la ligne supprimée.

```
def effacer_ligne(grille, emplacement_bloque):
    inc = 0
    #on parcourt la grille ligne par ligne en commençant par le bas
    for i in range(len(grille)-1, -1, -1):
        ligne = grille[i]
        #si (0,0,0) = case noire, n'apparait pas dans la ligne
        if (0,0,0) not in ligne:
            inc += 1
            #on mémorise le numéro de la ligne à effacer
            ind = i
            #On efface toutes les cases bloquées de cette ligne
            for j in range(len(ligne)):
                try:
                    del emplacement_bloque [(j,i)]
                except:
                    continue
     #s'il y a des lignes à effacer
    if inc > 0:
        #On trie les clés du dictionnaire emplacement_bloque
        #par ligne puis on inverse l'ordre de ces clés.
        #list(d) sur un dicionnaire d renvoie une liste de toutes les clés utilisées dans
        #le dictionnaire, dans l'ordre d'insertion. Dans emplacement_bloque les clés
        #sont (colonne, ligne)
        for key in sorted(list(emplacement_bloque), key=lambda x: x[1])[::-1]:
            colonne, ligne = key
            if ligne < ind:
                #on prend une ligne dans emplacement_bloque
                #et on la remet dans la liste à un index différent
                newKey = (colonne, ligne+ inc)
```

```
            emplacement_bloque [newKey] = emplacement_bloque.pop(key)
    return inc
```

Test

Ajouter une ligne dans la boucle de jeu lorsqu'on change de piece

```
#Changer de piece
    if changer_piece:
        #Mémoriser les cases occupées par la piece bloquée
        for pos in position_piece:
            p = (pos[0], pos[1])
            emplacement_bloque[p] = piece_enCours.couleur
        #choisr une autre pièce
        piece_enCours = choix_piece()
        changer_piece = False
        effacer_ligne(grille, emplacement_bloque)
```

Pour tester efficassement, on peut ralentir la chute des pièces en modifiant le chronomètre :

```
#créer un chronomètre qui déclenche l'évènement toute les 2 secondes
pygame.time.set_timer(CHUTE_PIECE, 2000)
```

et utiliser la touche « vers le bas » pour guider plus rapidement la pièce vers sont emplacement.