Tetris-Copy1

February 4, 2019

1 Programmer le jeu Tetris avec un SenseHat

1.1 Introduction

Le jeu Tetris est l'un des tout premiers jeux créés. Il s'agit de briques de différentes formes et longueurs qui descendent toute seule. Le but du jeu est de les empilés de manière méthodique pour former une ligne complète, qui disparaitra. Et ainsi de suite jusqu'à ce qu'une forme soit monté trop haut et ait bloqué l'apparition de la prochaine forme.

1.1.1 Importation des modules

Nous avons tout d'abord importer les modules utiles à notre jeu. Le module `gamelib` contient toutes les couleurs de base.

```
In [6]: from sense_hat import SenseHat
        from time import sleep, time
        from random import randint, choice
        from gamelib import *
```

1.1.2 Définition des variables propres à notre jeu

Voici les variables propres à un jeu comme Tetris ainsi que leur description.

```
In [26]: sense = SenseHat()
         color = (0, 0, 0) # Définition de la variable color, qui s'adaptera en fonction de la
         score = 0 # Le score total du joueur (le nombre de ligne complétées)
         state = 1 # State = 0 quand le jeu est en statique (entre les apparitions de formes)
         game = 1 # Game = 1 quand le jeu continue sinon c'est game over
         dx = 0 # Déplacement sur l'axe X (vers la gauche ou la droite)
         dy = 0 # Déplacement sur l'axe Y (vers le bas)
         dt = 1 # Temps entre chaque descente de forme en seconde
```

1.1.3 Définition des formes dans des matrices

L'écran de LED 8x8 étant assez restreint, nous avons décidés de réduire légérement les formes classiques de tetris en 3 formes principales. Ces matrices pourront être retournées pour faire une rotation de la forme par la suite. Dans les matrices, les 1 représentent les formes, donc les pixels allumés.

```
In [8]: # La barre verticale de 3 en cyan
        I = [[0, 1, 0],
             [0, 1, 0],
             [0, 1, 0]]

        # Le L en rouge
        L = [[1, 0],
             [1, 1]]

        # Le carré 2x2 en jaune
        o = [[1, 1],
             [1, 1]]

        shapes = (I, L, o) # Liste contenant les 3 formes

        P = choice(shapes) # Choisit une forme au hasard parmi les trois et la stock dans la v

        if P == L: # Assigne la couleur de la forme choisie à la variable color
            color = RED
        elif P == I:
            color = CYAN
        else:
            color = YELLOW
```

1.1.4 Définition des fonctions

Faire apparaître la forme Nous arrivons dans la partie la plus compliquée, tout d'abord nous avons créé une fonction qui fait apparaître une forme au hasard en haut au milieu du SenseHat.

```
In [9]: def print_matrix(M):
            """ Print the tetris shape at the top in the middle. """
            n = len(M)
            for y in range(n):
                for x in range(n):
                    if M[y][x] == 1:
                        sense.set_pixel(3+x, y, color)
```

Faire descendre la forme Il nous faut ensuite définir une fonction qui permet à la forme de descendre d'un pixel vers le bas. Cette fonction s'activerai donc toute les secondes automatiquement pour faire descendre la forme. On peut aussi l'activer manuellement pour descendre plus rapidement. Mais avant de faire réapparaître la matrice un cran en dessous, il faut d'abord la faire disparaître où elle est à l'instant.

```
In [10]: def delete_matrix_when_down(M, dx, dy):
             """ Delete the actual shape when it's moving down. """
             n = len(M)
             for y in range(n):  # Set every pixel of the matrix black
                     for x in range(n):
```

```
                if 0 <= y+dy <= 7:
                    if M[y][x] == 1:
                        sense.set_pixel(3+x+dx, y+dy, BLACK)
                elif M == [[0, 0, 0], [1, 1, 1], [0, 0, 0]] and y+dy == 8:  # Delete
                    if M[y][x] == 1:
                        sense.set_pixel(3+x+dx, y+dy, BLACK)
                else:
                    dy -= 1  # If it can't delete the shape, we put dy at same value
```

Après avoir défini la fonction qui supprime la matrice quand elle va vers le vas, créons la fonction générale qui fait descendre la forme d'un cran vers le bas.

```
In [11]: def print_matrix_down(M):
             """ Move shape down for 1 tile. """
             global dx, dy
             dy += 1  # Move the matrix one tile downward
             n = len(M)
             delete_matrix_when_down(M, dx, dy-1)
             for y in range(n):  # Print new matrix one tile downward
                 for x in range(n):
                     if 0 <= y+dy <= 7:
                         if M[y][x] == 1:
                             sense.set_pixel(3+x+dx, y+dy, color)
                     elif M == [[0, 0, 0], [1, 1, 1], [0, 0, 0]] and y+dy == 8:
                         if M[y][x] == 1:
                             sense.set_pixel(3+x+dx, y+dy, color)
                     else:
                         state = 0
```

On remarque une chose importante. La forme 'I' est un cas particulier à traiter en parallèle des deux autres formes. En effet si le 'I' est tourné en horizontal, la dernière ligne de la matrice est vide donc on peut descendre d'une case de plus. Nous retrouverons ce cas assez souvent dans les futurs fonctions.

Déplacer la forme à gauche ou à droite Mettons maintenant en place des fonctions qui permettent à la matrice de se déplacer à gauche ou à droite. Ces fonctions seront par la suite activées par le joystick. Malheureusement, les fonctions qui suivent sont compliquées à optimiser à l'aide de sous-fonctions. Ce qui explique la longueur de cette dernière.

```
In [12]: def print_matrix_left(M):
             """ Move shape left for 1 tile. """
             global dx, dy
             dx -= 1
             n = len(M)
             for y in range(n):  # Stop the shape when it's against another one
                 if M == [[0, 1, 0], [0, 1, 0], [0, 1, 0]] and -1 <= 3+dx <= 7:
                     if M[y][1] == 1 and sense.get_pixel(3+dx+1, y+dy)!=[0, 0, 0]:
                         dx += 1
```

```
                    return ;
            elif 0 <= 3+dx <= 7:
                if M[y][0] == 1 and sense.get_pixel(3+dx, y+dy) != [0, 0, 0]:
                    dx += 1
                    return ;

        for y in range(n):  # Set the pixel of the actual matrix to black (delete it)
            for x in range(n):
                if 0 <= 3+x+dx <= 7:
                    if M[y][x] == 1:
                        sense.set_pixel(3+x+dx+1, y+dy, BLACK)
                elif M == [[0, 1, 0], [0, 1, 0], [0, 1, 0]] and 3+x+dx == -1: # If the 'I
                    if M[y][x] == 1:
                        sense.set_pixel(3+x+dx+1, y+dy, BLACK)
                else:
                    dx +=  1
        print_new_matix_when_left(M, dx, dy, n)
```

```
In [13]: def print_new_matix_when_left(M, dx, dy, n):
             """ Print a new matrix moved to the left. """
             for y in range(n):  # Print the new matrix one tile leftward
                 for x in range(n):
                     if 0 <= 3+x+dx <= 7:
                         if M[y][x] == 1:
                             sense.set_pixel(3+x+dx, y+dy, color)
                     elif M == [[0, 1, 0], [0, 1, 0], [0, 1, 0]] and 3+x+dx == -1:
                         if M[y][x] == 1:
                             sense.set_pixel(3+x+dx, y+dy, color)
```

Nous retrouvons ici à peu près la même architecture que la fonction précédente, on supprime la forme originelle et on affiche la nouvelle forme décalée. En prenant en compte le cas particulier encore une fois de la barre, qui si elle est droite peut se déplacer d'une case supplémentaire sur les côtés.

Nous avons ici un problème supplémentaire ; si la forme en rencontre une autre, elle ne peut plus avancer. Il a donc fallu déterminer si des pixels allumés gênaient le passage pour savoir si on pouvait décaler la forme.

Note : Nous remarquons qu'une fonction `delete_matrix` *générale pour toute les fonctions auraient été pratiques. Hélàs ce n'est pas si simple, cela nous aurait donné une fonction fort pratique mais très longue, ç'aurait été plus problématique que bénéfique. C'est d'ailleurs pour la même raison que nous n'avons pas pu créer de fonction* `print_matrix` *générale.*

Voici ensuite la fonction qui déplace la forme vers la droite cette fois-ci. Elle est quasiment identique à `print_matrix_left` à quelques détails près.

```
In [14]: def print_matrix_right(M):
             """ Move shape right for 1 tile. """
             global dx, dy
             dx += 1
             n = len(M)
```

```
            for y in range(n):  # Stop the shape when it's against another one
                if M == [[0, 1, 0], [0, 1, 0], [0, 1, 0]] and -1 <= 3+dx <= 6:
                    if M[y][1] == 1 and sense.get_pixel(3+dx+1, y+dy) != [0, 0, 0]:
                        dx -= 1
                        return ;
                if M == [[0, 0, 0], [1, 1, 1], [0, 0, 0]] and -1 <= 3+dx <= 5:
                    if M[y][2] == 1 and sense.get_pixel(3+dx+2, y+dy) != [0, 0, 0]:
                        dx -= 1
                        return ;
                elif 0 <= 3+dx <= 6:
                    if M[y][1] == 1 and sense.get_pixel(3+dx+1, y+dy) != [0, 0, 0]:
                        dx -= 1
                        return ;

            for y in range(n):  # Set the pixel of the actual matrix to black (delete it)
                for x in range(n):
                    if 0 <= 3+x+dx <= 7:
                        if M[y][x] == 1:
                            sense.set_pixel(3+x+dx-1, y+dy, BLACK)
                    elif M == [[0, 1, 0], [0, 1, 0], [0, 1, 0]] and 3+x+dx == 8: # If the 'I'
                        if M[y][x] == 1:
                            sense.set_pixel(3+x+dx-1, y+dy, BLACK)
                    else:
                        dx -= 1
            print_new_matix_when_right(M, dx, dy, n)

In [15]: def print_new_matix_when_right(M, dx, dy, n):
            """ Print a new matrix moved to the right. """
            for y in range(n):  # Print the new matrix one tile rightward
                for x in range(n):
                    if 0 <= 3+x+dx <= 7:
                        if M[y][x] == 1:
                            sense.set_pixel(3+x+dx, y+dy, color)
                    elif M == [[0, 1, 0], [0, 1, 0], [0, 1, 0]] and 3+x+dx == 8:
                        if M[y][x] == 1:
                            sense.set_pixel(3+x+dx, y+dy, color)
```

Faire tourner la forme sur elle-même Finalement, terminons avec la fonction qui permet de faire tourner les formes. Nous l'avons appelée `rotate_90` car elle exerce une rotation de 90ř de la matrice vers la droite. Nous tenons à dire que nous avons essayé de réaliser cette fonction par nous-même pendant longtemps sans succès. Nous avons donc cherché une fonction capable de faire cela sur internet. Une partie de cette fonction (la boucle for) est donc empruntée et n'est pas de nous.

```
In [20]: def print_rotated_matrix(n, matrix):
            """ Print the matrix after the rotation. """
            for y in range(n):
                for x in range(n):
```

```
                    if matrix[y][x] == 1:
                        sense.set_pixel(3+x+dx, y+dy, color)
```

```
In [21]: def rotate_90(matrix):
             """ Turn the shape 90 degrees right. """
             delete_matrix_when_down(matrix, dx, dy)
             n = len(matrix)
             for layer in range((n + 1) // 2):  # Rotate the matrix (borrowed on internet)
                 for index in range(layer, n-1-layer, 1):
                     matrix[layer][index], matrix[n-1-index][layer], \
                         matrix[index][n-1-layer], matrix[n-1-layer][n-1-index] = \
                         matrix[n-1-index][layer], matrix[n-1-layer][n-1-index], \
                         matrix[layer][index], matrix[index][n-1-layer]
             print_rotated_matrix(n, matrix)
             return matrix
```

Ici, contrairement aux fonctions `print_matrix_left` et `print_matrix_right`, nous pouvons utiliser la fonction de suppression de la matrice quand on descend. Car cette dernière ne prend en compte aucun cas particulier (une forme est présente sur la gauche/droite, la forme est au bord etc ...). La fonction `print_rotated_matrix` fonctionne exactement de la même manière que les autres.

Supprimer les lignes remplies et incrémenter le score Lorsqu'une forme est déposée, le jeu doit vérifier tout les lignes de pixels, si une ligne est remplie de pixels, alors il la supprime et fait descendre de 1 toute les lignes du dessus. Il faut donc vérifier grâce aux boucles `for` toute la grille.

```
In [22]: def check_if_lines_are_completed():
             """ Check if lines are completed to delete them and increase the score. """
             global score
             for g in range(8):
                         for i in range(8):
                             a = 0
                             for j in range(8):
                                 if sense.get_pixel(7-j, 7-i) != [0, 0, 0]:
                                     a += 1
                                     if a == 8:
                                         score += 1 # Inscrease the score if the line is compl
                                         delete_lines(i)
```

Après avoir vérifier chaque ligne, s'il y a effectivement une ligne de remplie, nous appelons la fonction `delete_lines` et insérons comme argument le numéro de la ligne à supprimer.

```
In [23]: def delete_lines(i):
             """ Delete completed lines and move every line above one tile downward. """
             for k in range(8):
                 sense.set_pixel(k, 7-i, BLACK)
             for c in reversed(range(7-i)):
                 for d in range(8):
                     sense.set_pixel(d, c+1, (sense.get_pixel(d, c)))
                     sense.set_pixel(d, c, BLACK)
```

1.1.5 Les corps du jeu dans une fonction `main()`

Nous entrons dans le coeur du jeu. Avant de développer ce dernier, il faut savoir que pour éviter de mettre 70 lignes de code d'un coup nous l'avons divisé en plusieurs parties. Il faut néanmoins savoir que toute les prochaines parties expliquées sont imbriquées dans une grande fonction `main()` qui permet de recommencer le jeu si on le souhaite.

Tout d'abord, remettons toute les variables du jeu à zéro et faisons apparaître la première forme.

```
In [24]: def main():
             """ The core of the game. """
             global x, y, dx, dy, color

             P = choice(shapes)

             if P == L:
                 color = RED
             elif P == I:
                 color = CYAN
             else:
                 color = YELLOW

             t0 = time()

             game = 1

             state = 1

             print_matrix(P)
```

État dynamique (`state = 1`) Juste avant cette étape, nous avons déjà fait apparaître une forme de départ, la variable state commence donc en état dynamique. Le jeu est donc mis par défaut en `state = 1` qui veut dire 'dynamique', la forme peuvent donc descendre.

Le joystick La variable `game` permet au jeu de faire spawn des formes à l'infini jusqu'à ce que le joueur perde et passe la variable en zéro.

```
In [ ]:     while game == 1:
                while state == 1:
                    for event in sense.stick.get_events():
                         if event.action == 'pressed' and event.direction == 'middle':
                            if P == [[0, 1, 0], [0, 1, 0], [0, 1, 0]]:
                                if dx == -4 or dx == 3:
                                    pass  # The vertical 'I' can not rotate when it's in the r
                                elif sense.get_pixel(3+dx, dy+1) != [0, 0, 0] or sense.get_pixe
                                    pass
                                else:
                                    rotate_90(P)
```

```
            elif P == [[0, 0, 0], [1, 1, 1], [0, 0, 0]]:
                if dy == 6:
                    pass  # The horizontal 'I' can not rotate when it's at the
                else:
                    rotate_90(P)
            else:
                rotate_90(P)
        elif event.direction == 'down' and event.action == 'pressed':
            print_matrix_down(P)
        elif event.direction == 'left' and event.action == 'pressed':
            print_matrix_left(P)
        elif event.direction == 'right' and event.action == 'pressed':
            print_matrix_right(P)
        elif event.direction == 'up' and event.action == 'held': #If the play
             game = 0
             state = 0
```

Il s'agit ici de vérifier si le joueur appuie sur le joystick. * Si le bouton est appuyé, il tourne la forme à l'aide de la fonction `rotate_90`, tout en traitant le cas particulier du 'I'. Si ce dernier est vertical est dans le bord, il ne peut pas devenir un 'I' en horizontal. De même que si le 'I' est couché et tout en bas, il ne peut pas de retourner non plus. Ces cas sont donc écarté à l'aide de `pass` qui ne fait rien. * Le bouton du bas permet à la forme de descendre plus vite si le joueur le souhaite. * Le bouton de gauche et de droite sont affectés à leur fonction de déplacement respective. * Si le joueur désire quitter le jeu avant d'avoir fini, il suffit de rester appuyer avec le joystick vers le haut, le jeu quittera alors automatiquement.

Critère pour arrêter la forme et passer en mode statique (ou perdre) Il effectue ensuite un test qui permet de déterminer si la forme est tout en bas (`dy = 6 ou 5` en fonction de la forme). Si c'est le cas, on passe en mode statique (`state = 0`).

In [ ]:
```
if P == [[0, 0, 0], [1, 1, 1], [0, 0, 0]] and dy == 6:
    state=0
elif P == [[0, 1, 0], [0, 1, 0], [0, 1, 0]] and dy == 5:
    state=0
elif P == O and dy == 6:
    state=0
elif P == L and dy == 6:
    state=0
```

Mais le passage en mode statique n'est pas seulement définit par le fait que la forme touche le sol. La forme doit aussi s'arrêter si elle touche une forme en dessous d'elle.

In [ ]:
```
n=len(P)
for x in range(n):  # Check if the shape can drop
    if P == [[0, 1, 0], [0, 1, 0], [0, 1, 0]] and dy < 5:
        if sense.get_pixel(dx+4, dy+3) != [0, 0, 0]:
            if dy == 0:
                game = 0
```

```
                state = 0
            elif dy < 6 and P != [[0, 1, 0], [0, 1, 0], [0, 1, 0]]:
                if P[1][x] == 1 and sense.get_pixel(x+dx+3, dy+2) != [0, 0, 0]:
                    if dy == 0:  # If the shape is blocked just after spawning, ga
                        game = 0
                    state = 0
```

Le cas particulier du L Hélàs se présente à nous un autre problème. Non pas concernant le 'I' mais le 'L' cette fois-ci. Le 'L' est la seule forme concave de notre liste, il peut donc s'imbriquer dans d'autre forme.

```
In [ ]:     Lcomplet = 0  # Check if the shape 'L' is completed by another external pi
            for x in range (n):
                for y in range(n):
                    if P == [[1, 1],[0, 1]] or P == [[1, 1],[1, 0]]:
                        if sense.get_pixel(x+dx+3, y+dy) != [0, 0, 0]:
                            Lcomplet += 1
                        if Lcomplet == 4:
                            state = 0
```

Nous avons donc procédé de cette manière ; le 'L' s'arrête si un pixel du dessous est autre que noir (bloc de code précédent) OU il s'arrête si les quatre cases de la matrice du L sont remplies.

La descente automatique des formes Le dernier point à traiter est trivial mais l'un des plus important, c'est la descente à chaque `dt` seconde. Nous avons mis en place un test de variable plutôt qu'un `sleep(1)`. Car ce dernier aurait paralysé pendant 1 seconde toute la boucle du state 1 au lieu de continuer en boucle. Cela aurait par exemple eu comme conséquence le fait de ne pouvoir appuyer que sur un bouton par seconde sur le joystick.

```
In [ ]:     t = time()
            if t > t0 + dt:
                matrix_print_down(P)
                t0 = t
```

État statique (`state = 0`)

Supprimer les lignes remplies La première chose à vérifier lorsqu'une forme a été placée est si une ligne est remplie. Si oui, il la supprime et descend d'un pixel tout ce qu'il y a au dessus. Il se charge également de remettre les variables de déplacement, de temps à zéro. Il fait ensuite apparaître la prochaine forme pour le prochain round.

```
In [ ]:     while state == 0:

                check_if_lines_are_completed()

                P = choice(shapes)  # Pick randomly a shape for one round

                if P == L:
```

```
        color = RED
    elif P == I:
        color = CYAN
    else:
        color = YELLOW

    t0 = time()

    dx = 0  # Set back the position where we print the matrix to original sett
    dy = 0

    print_matrix(P)
```

Game over ? Et affichage du score final Avant de relancer le jeu en dynamique, il faut vérifier que game ne soit pas égal à 0, ce qui voudrait dire un game over. La seule manière que game ait été assigné à la valeur 0 se situe dans le critère pour arrêter la forme. Si la forme doit s'arrêter juste après être apparue, c'est game over. (Voir *Critère pour arrêter la forme et passer en mode statique*)

Lorsque le joueur a perdu, un message 'Game Over' s'affiche et lui montre son score.

```
In [ ]:    if game == 0:  # When game is over, displays the score
                sense.show_message('Game over ! Score :', scroll_speed=0.05
                sense.show_message(str(score), scroll_speed=0.2)
                t0 = time()
                active = True
                while active: # Play again
                    sense.show_letter('?')
                    for event in sense.stick.get_events():
                        if event.direction == 'middle' and event.action ==
                            game = 1
                            active = False
                            sense.clear()
                            main()
                    t = time()
                    if t > t0 + 3: # After 3 seconds if the player did not
                        sense.clear()
                        return ;
           else:
                state = 1
```

Si le joueur à perdu, il a possiblité lorsqu'un '?' est affiché d'appuyer sur le bouton du milieu pour relancer une partie. S'il ne fait rien pendant 3 sec, le jeu s'arrête.

Le module L'un des buts de ce petit jeu est de le transformer en module. Pour le compiler avec d'autres jeux et faire un mini-centre de jeu-vidéo pour le SenseHat ! Voici donc le petit bout de code qu'il faut rajouter à la fin pour pouvoir lancer votre jeu à la séléction de son module.

```
In [ ]: # Execute the main() function when the file is executed,
        # but do not execute when the module is imported as a module.
```

```
print('module name =', __name__)

if __name__ == '__main__':
    main()
```

1.1.6 Points à améliorer

Notre jeu est plutôt complet et marche étonnemment bien, néanmoins il resterais, si nous avions plus de temps, quelques points à améliorer. Il est important de prendre du recul sur notre travail * Le plus important est le fait que nous voulions un temps d'une seconde de latence avant la réapparition de la forme. Mais lorsque nous mettions un sleep(1) en premier dans la boucle while state == 0, nous rencontrions un problème. Nous pouvions bouger la forme (y compris vers le bas) avant son apparition et par conséquent un peu tricher sur chaque apparition d'une nouvelle forme. Nous avons donc choisi d'abandonner ce petit côté nostalgique du temps de latence et faire directement apparaître la nouvelle forme. Ce qui rajoute une difficulté au jeu ! * Dû au manque de temps, nous aurions pu mieux optimiser notre code et enlever, je pense, beaucoup de lignes de code à l'ensemble. Mais lorsque le code fonctionne, il est difficile de prendre du recul et de le voir sous un autre point de vue.