UNIVERSITÉ
CAEN
NORMANDIE

Rapport Tetris Multijoueur

Sahin Tolga

15 avril 2019

Table des matières

1 Objectif du projet

1.1 Description du concept derrière l'application

Le concept du projet est de développer une interface graphique dans laquelle on fera tomber et bouger différentes pièces dans une grille. Lorsque une ligne de la grille est complète, on la réinitialise et augmente le score de jeu. Le jeu est une boucle de cette manière et cela jusqu'à ce que une pièce arrive à la hauteur de départ. Donc, brièvement on reprend le principe et les règles d'un tetris classique.

1.2 Ce qu'il fallait faire

Il fallait réaliser une interface graphique qui contient une grille dans laquelle des pièces seront générés aléatoirement. Ensuite, il fallait créer et gérer le jeu, par exemple les mouvements et les collisions afin de pouvoir y jouer.

1.3 Ce qui existe déjà

Les propriétés de l'interface graphique sont déjà existantes, donc il n'y avait qu'à les comprendre et les utiliser correctement. Les pièces de Tetris sont-elles aussi déjà connu, donc il a juste fallu les déclarer dans un langage par laquelle je pouvais les manipuler.

2 Fonctionnalités implémentées

2.1 Description des fonctionnalités

Mon jeu est une application Tkinter. Dans laquelle, j'ai insérer deux fonctionnalités. Tout d'abord, l'initialisation du jeu est ma première fonctionnalité.

La grille de jeu sera une liste de liste de N lignes et M colonnes. Que l'on initialise dès lorsque l'on clique sur le bouton fait pour "Initialiser".

Puis, l'implémentation des pièces dans la grille (et du jeu en général). Les pièces sont déclarées sous formes de liste de liste avec une numérotation binaire à l'intérieur en 5x5.

```
S = [[[0,0,0,1,1],
   [0,0,1,1,0],
   [0,0,0,0,0],
   [0,0,0,0,0],
   [0,0,0,0,0]],
  [[0,0,0,0,0],
   [0,0,0,1,0],
   [0,0,0,1,1],
   [0,0,0,0,1],
   [0,0,0,0,0]]]

Z =[[[0,0,1,1,0],
  [0,0,0,1,1],
  [0,0,0,0,0],
  [0,0,0,0,0],
  [0,0,0,0,0]],
 [[0,0,0,0,0],
  [0,0,0,1,0],
  [0,0,1,1,0],
  [0,0,1,0,0],
  [0,0,0,0,0]]]
```

L'initialisation de mon jeu se fait grâce aux différentes méthodes présentes dans ma classe et surtout dans laquelle je fais tourner une classe tick si le jeu n'est pas terminé. Elle permet le fonctionnement en continu du jeu jusqu'à ce que le joueur perd.

J'ai implémenter une pièce supplémentaire dans le jeu, que j'ai nommé V. Le score, et des niveaux qui permettront au jeu de devenir de plus en plus rapide.

Puis, j'ai développer une IA qui pour l'instant arrive seulement à se déplacer.

Enfin, j'ai intégrer à l'aide de la bibliothèque socket, une partie réseau à laquelle on peut jouer dans deux différents ordinateurs d'une manière synchronisée au Tetris. Si l'un des deux joueurs perd, l'autre joueur reçevra un avertissement.

2.2 Organisation du projet

Le projet s'organise dans un fichier python dans lequel j'ai décidé de développer sous forme d'application Tkinter. Pour ma part, j'ai fais le choix de n'utiliser qu'une seule classe pour mon programme. Afin que la grille de celui-ci soit plus facilement manipulable dans l'instance. Donc, tout mon programme est écrit dans une classe Tkinter qui se nomme Tetris.

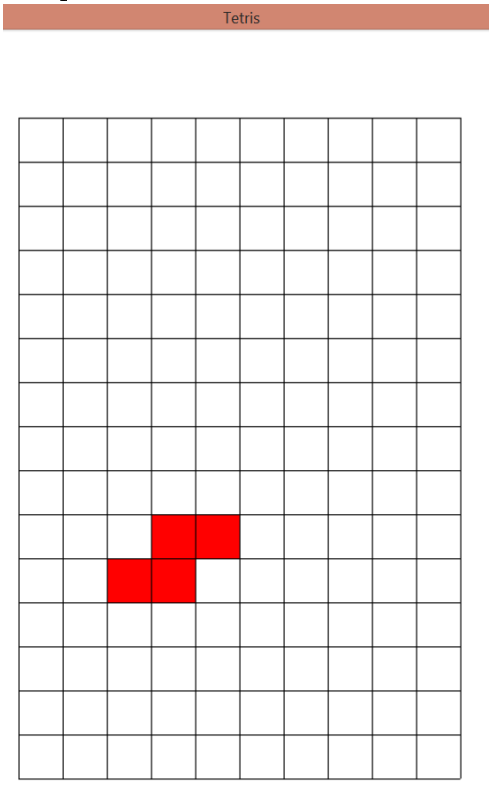
3 Élements techniques

3.1 Algorithmes

L'initialisation de la grille s'effectue grâce à une méthode initGrille, dans laquelle j'utilise d'autres méthodes.

```
def initGrille(self):
    self.can.delete(ALL)
    self.grille=[[0 for x in range(Tetris.NBCOL)] for x in range(Tetris.NBLIG)]
    #self.tirePiece()
    self.spawn()
    for ligne in range(Tetris.NBLIG+1):
        self.can.create_line(Tetris.MARGE,Tetris.MARGE+ligne*Tetris.TAILLE_CASE,Tetris.WIDTH+Tetris.MARGE,Tetris.MARGE+ligne*Tetris.TAILLE_CASE)
    for col in range(Tetris.NBCOL+1):
        self.can.create_line(Tetris.MARGE+col*Tetris.TAILLE_CASE,Tetris.MARGE,Tetris.MARGE+col*Tetris.TAILLE_CASE,Tetris.HEIGHT+Tetris.MARGE)
    self.fillGrille()
    self.tick()
    print(self.indexBasPiece())
```

Ce qui nous donne un résultat de ce type :



Le précèdent algorithme contient notamment une méthode tick qui lui aussi est un algorithme principal de mon programme. Il permet la descente des pièces, et est en quelque sorte c'est l'algorithme de base du jeu car c'est lui qui nous indiquera lorsque la partie est terminé.

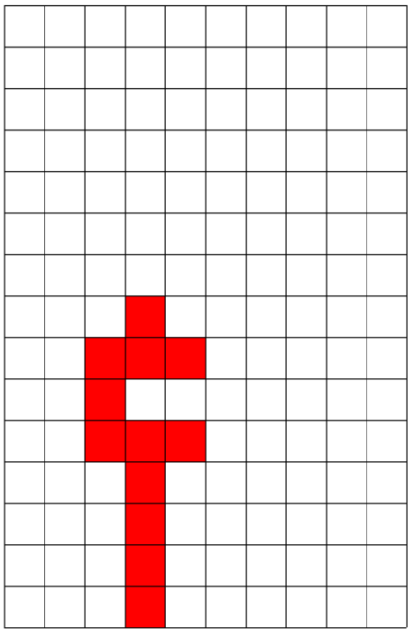
Voici le code de cette algorithme :

```
def tick(self):
    tick = self.after(Tetris.VITESSE, self.tick)
    if self.isBlocked() :
        for lig in range(Tetris.NBLIG):
            for col in range(Tetris.NBCOL):
                if self.grille[lig][col]==1:self.grille[lig][col]=2
        if not self.fini():
            self.tirePiece()
            self.spawn()
    else:self.pieceTombe()
```

Puis j'ai fais en sorte que lorsqu'une pièce est définitivement posé, donc soit elle est rentré en collision soit elle est arrivé en bas. Alors sa valeur prendra 2. Ce qui m'a permis de créer des conditions (utiliser dans mes algorithmes) qui vont permettre au jeu de bien se dérouler. Donc j'ai pu gérer en sorte les collisions de manière à ce que le programme sache gérer et voir quel est la pièce à la ligne d'après.

```
def isBlocked(self):
    if 1 in self.grille[-1]: return True
    for lig in range(Tetris.NBLIG-1):
        for col in range(Tetris.NBCOL):
            if self.grille[lig][col]==1 and self.grille[lig+1][col]==2:
                return True
    return False
```

Ce qui permettra de faire ceci :



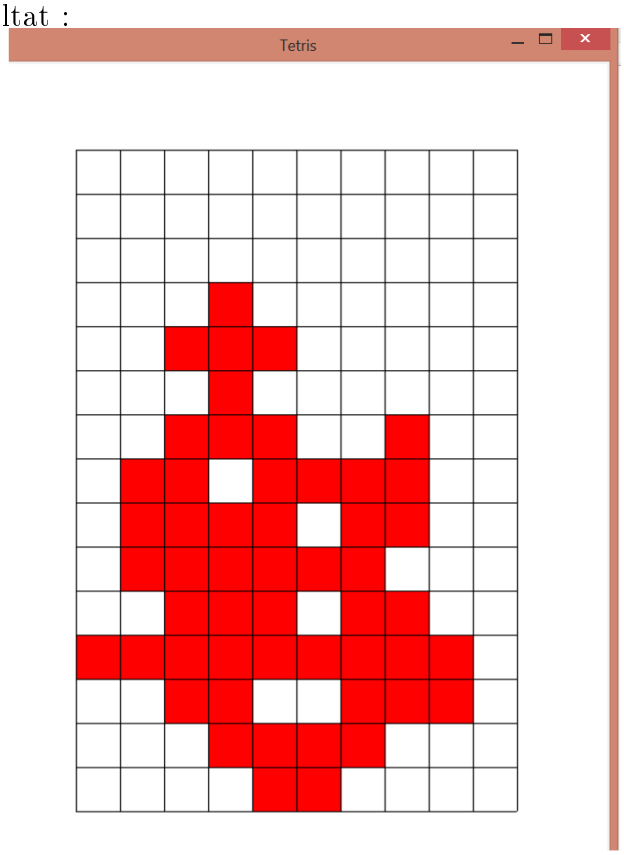
Donc c'est une condition qui est réutilisable, donc notamment dans la méthode tick mais aussi dans mes mouvements. Dans laquelle je vérifie toujours cette condition mais aussi une nouvelle en fonction du mouvement latéral qui vérifiera la colonne +1 ou la colonne -1 de la grille. Plus simplement, on vérifie si il n'y a pas de pièce déjà fixé dans les colonnes d'après ou d'avant.

```
def cantMovedroite(self):
    for lig in range(Tetris.NBLIG)[::-1]:
        for col in range(Tetris.NBCOL):
            if self.grille[lig][col]==1 and self.grille[lig][col+1]==2:
                return True
    return False

def cantMovegauche(self):
    for lig in range(Tetris.NBLIG)[::-1]:
        for col in range(Tetris.NBCOL):
            if self.grille[lig][col]==1 and (self.grille[lig][col-1]==2 or col==0):
                return True
    return False
```

combiné aux méthodes :

```
#mouvement à gauche
    def lateralgauche(self,event):
        if not self.cantMovegauche():
            for lig in range(Tetris.NBLIG)[::-1]:
                for col in range(Tetris.NBCOL):
                    if self.grille[lig][col]==1:
                        self.grille[lig][col]=0
                        self.grille[lig][col-1]=1

#mouvement à droite
    def lateraldroit(self,event):
        try:
            if not self.cantMovedroite():
                for lig in range(Tetris.NBLIG)[::-1]:
                    for col in range(Tetris.NBCOL)[::-1]:
                        if self.grille[lig][col]==1:
                            self.grille[lig][col]=0
                            self.grille[lig][col +1]=1
        except IndexError:
            pass # car on sera forcément en indexerror si o
```

Va nous permettre d'avoir le tetris de base tout à faire jouable. Donc en voici un résultat :



3.2 Structures de données

Mes données sont structurées principalement par des grilles (des listes de listes). Donc je peux reprendre l'exemple de mes pièces :

```
S = [[[0,0,0,1,1],
   [0,0,1,1,0],
   [0,0,0,0,0],
   [0,0,0,0,0],
   [0,0,0,0,0]],
  [[0,0,0,0,0],
   [0,0,0,1,0],
   [0,0,0,1,1],
   [0,0,0,0,1],
   [0,0,0,0,0]]]

Z =[[[0,0,1,1,0],
  [0,0,0,1,1],
  [0,0,0,0,0],
  [0,0,0,0,0],
  [0,0,0,0,0]],
[[0,0,0,0,0],
  [0,0,0,1,0],
  [0,0,1,1,0],
  [0,0,1,0,0],
  [0,0,0,0,0]]]
```

La grille se construit avec des boucles itératives de cette manière :

```
self.grille=[[0 for x in range(Tetris.NBCOL)] for x in range(Tetris.NBLIG)]
```

L'initialisation de notre classe se présente comme ceci :

```
def __init__(self):
    Tk.__init__(self)
    self.title("Tetris")
    self.grille = None
    self.piece = Tetris.pieces[0]
    self.ori=0
    self.score = 0
    self.can = Canvas(self, width=Tetris.WIDTH+2*Tetris.MARGE, height=Tetris.HEIGHT+2*Tetris.MARGE,bg="white"
    self.can.pack()
    self.can.create_text(Tetris.WIDTH//2,Tetris.HEIGHT//2,text="Bienvenu sur mon tetris")
    Button(self,text="Jouer",command=self.initGrille).pack(side=LEFT)
    Button(self,text="Test",command=self.pieceTombe).pack(side=LEFT)
    Button(self,text="Quitter",command=self.quitter).pack()
    self.bind("<Left>", self.lateralgauche)
    self.bind("<Right>", self.lateraldroit)
    self.bind("<Up>", self.rotate)
```

Les variables de la classe sont :

```
TAILLE_CASE = 40
NBCOL = 10
NBLIG = 15
MARGE=80
WIDTH = NBCOL*TAILLE_CASE
HEIGHT = NBLIG*TAILLE_CASE
VITESSE= 200                    pieces = [S,Z,I,O,J,L,T]
```

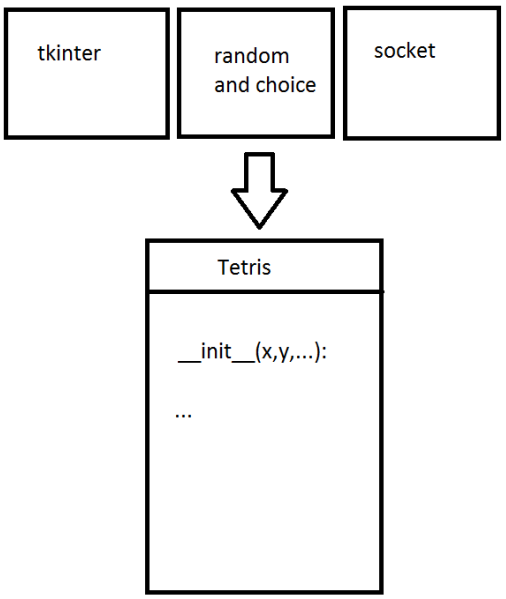
3.3 Bibliothèque

Mes bibliothèques utilisés sont Tkinter : pour toute la gestion de l'interface graphique. Les bibliothèques random et choice qui m'ont permis d'utiliser la gestion aléatoire. Enfin, j'envisage d'utiliser socket pour toutes les manipulations réseaux du programme.

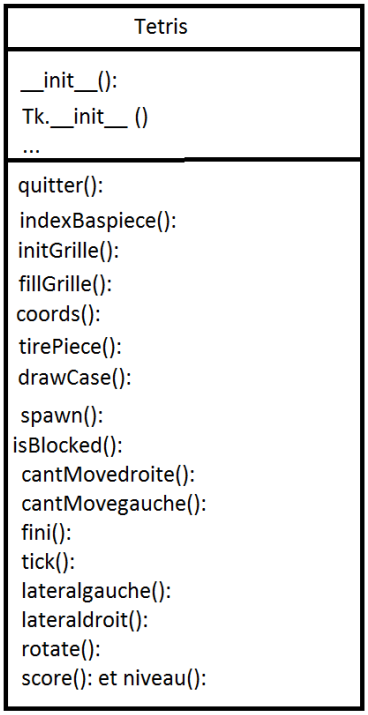
4 Architecture du projet

4.1 Diagrammes des modules et des classes

Mon projet est une application Tkinter. Voici un diagramme des modules.



Et voici le diagramme qui résume la création de ma classe :



5 Expérimentations et usages

5.1 Essais

J'ai tout d'abord experimenter la création d'un Tetris sous sa forme alphanumérique. Dans le but de pouvoir bien voir clairement le côté algorithmique du jeu.

J'avais d'abord réalisé une première version sous Pygame et avec des fonctions, au niveau des fonctions je n'avais pas réellement rencontré de difficulté mais j'ai souhaité passer au classes par simple curiosité. Tandis que pour Pygame, sa bibliothèque me parlait moins bien que celle de Tkinter donc c'est la raison pour laquelle j'ai décidé de changer.

5.2 Capture d'écran

Voici une capture d'écran d'une de mes premières versions : qui été fait en alphanumérique.

```
def creePiece(nom=L,ori=0,l=0,c=4):
    for i in range(l,l+4):
        grille[i][c:c+4]=nom[ori][i-l]

def efface(l,c):
    grille[l][c:c+4]=[0,0,0,0]

def affiche():
    print("\n")
    for ligne in grille:
        res=""
        for e in ligne:
            if e==0:res+='.'
            if e==1:res+='*'
        print(res)

def suivant():
   global lig
   lig+=1
   creePiece(l=lig,c=col,ori=ori)
   efface(lig-1,col)

def tourner():
    global ori
    ori=(ori+1)%4

#il ne faut pas d'autres 1 lors de la rotation
def verification():
    pass
```

5.3 Mesure de performance

J'ai mesurer avec le gestionnaire de tâche, l'application a de bonnes performances sans que je n'ai rien eu à y faire. Donc, j'ai juste bloquer les FPS à 30 afin que ceci reste stable et que le jeu soit fluide.

6 Conclusion

6.1 Récapitulatif des fonctionnalités principales

Mon projet contient toutes les fonctionnalités que peut avoir un Tetris basique, mais en supplément j'implémente un côté multijoueur, mais aussi quelques spécificités comme une pièce supplémentaire, des niveaux et un score.

6.2 Propositions d'améliorations

Il faudrait améliorer l'aspect graphique de mon application. Le côté multijoueur est lui aussi à améliorer, donc l'IA mais aussi la gestion réseau. Il y'a aussi un léger bug sur le haut de ma grille que je n'ai pas réussi à résoudre. Donc au moment où j'écris ce sont les améliorations à clairement effectuer, je vais faire de mon mieux pour que jusqu'à la soutenance le projet soit plus avancé qu'il l'est actuellement et donc améliorer tout les points cités précédemment.