Documentation Complète Tetris

Pierre MAGIEU - TD4

Fichier modele.py

```
class ModeleTetris:
    def __init__ (self, nb_lig=24, nb_col=14, base = 4):
        self.__haut = nb_lig
        self.__larg = nb_col
        self.__base = base
        self.__score = 0
        self.__terrain:list = [[-2 for _ in range(self.__larg)] for _ in range(self.__base)] + [[-1 for _ in range(self.__larg)] for _
        self.__forme = Forme(self)
        self.__suivante = Forme(self)
        self.__commence = False
        self.__pause = False
```

Initialisation de la classe ModeleTetris où le terrain est une matrice, c'est à dire une liste de liste composé de chiffres compris entre -2 et 6. C'est chiffres vont correspondre aux valeurs des différentes case qui définissent la couleur de ces dernières.

```
def get_largeur(self):
        '''ModeleTetris -> int
        Récupère la largeur de la classe ModeleTetris
        '''
        return self.__larg

def get_hauteur(self):
        '''ModeleTetris -> int
        Récupère la hauteur de la classe ModeleTetris
        '''
        return self.__haut
```

Ces deux méthodes vont simplement servir à récupérer la largeur et la hauteur de la classe ModeleTetris depuis n'importe quel fichier afin de vérifier si la pièce est arriver en bas du terrain par exemple.

```
def get_valeur(self,lig,col):
        '''ModeleTetris, int, int-> int
        Récupère la valeur d'une case / le chiffre associé à tel élément de telle liste
        '''
        return self.__terrain[lig][col]
```

Cette méthode permet de récupérer la valeur d'une case donc le chiffre associé à tel élément de telle liste.

```
def est_occupe(self, lig, col):
        '''ModeleTetris, int, int -> Bool
        Vérifie si une case existe avec ses coordonnées
        Vérifie si la case est vide grâce à sa valeur associée
        '''
        if lig >= self.get_hauteur() or col >= self.get_largeur() or lig < 0 or col < 0:
            return True
        if self.__terrain[lig][col] >= 0:
            return True
        return False
```

est_occupe() va vérifier si une case du terrain est occupé, c'est à dire si la valeur de l'élément est supérieure ou égale à 0. Ceci s'explique par le fait que les formes sont composées d'un chiffre compris entre 0 et 6.

```
def fini(self):
        '''ModeleTetris -> Bool
        Vérifie si le jeu doit s'arrêter
        '''
        for elmt in self.__terrain[self.__base-1]:
            if elmt != -2:
                self.__commence = 0
                return True
        return False
```

Cette méthode permet de vérifier si le jeu doit s'arrêter. Dans cette étape le jeu se termine seulement si la forme atteint les cases grises, la base du terrain. Donc si les listes qui compose la base contiennent autre chose que -2, le jeu est fini et le code s'arrête.

```
def ajoute_forme(self):
        '''ModeleTetris -> None
        Permet de changer la valeur d'une case, sa couleur sur le terrain
        '''
        l = self.__forme.get_coords()
        for elmt in l:
            x,y = elmt
            self.__terrain[y][x] = self.__forme.get_couleur()
```

La méthode ajoute_forme() va s'occuper de changer les valeurs des cases, c'est à dire changer leur couleur sur le terrain.

```
def forme_tombe(self):
        '''ModeleTetris -> Bool
        Permet de dessiner la forme sur le terrain et d'en créer une nouvelle
        quand la précédente ne tombe plus
        '''
        if self.__forme.tombe():
            self.ajoute_forme()
            self.__forme=self.__suivante
            self.__suivante=Forme(self)
            self.supprime_lignes_completes()
            return True
        return False
```

Cette méthode permet créer une nouvelle forme après avoir dessiner la précédente une fois que celle-ci à fini de tomber.

```
def get_couleur_forme(self):
        '''ModeleTetris -> Int
        Récupère la valeur associé à la forme, sa couleur
        '''
        return self.__forme.get_couleur()
```

La méthode suivante permet de récupérer la couleur de la forme, la valeur qui y est associée.

```
def get_coords_forme(self):
        '''ModeleTetris -> tuple(Int, Int)
        Récupère les coordonnées d'une forme
        '''
        return self.__forme.get_coords()

def get_terrain(self):
        '''ModeleTetris -> List(List(Int))
        Récupère la liste qui compose le terrain
        '''
        return self.__terrain
```

Les deux méthodes suivantes servent à récupérer des informations liées à la forme qui sont les positions des cases qui la compose dans sur le terrain et la deuxième permet d'utiliser la liste self.__terrain dans d'autre fichier.

```
def forme_a_gauche(self):
        '''ModeleTetris -> None
        Fait se diriger la forme vers la gauche
        '''
        if self.__commence:
            if not self.__pause:
                self.__forme.a_gauche()

def forme_a_droite(self):
        '''ModeleTetris -> None
        Fait se diriger la forme vers la droite
        '''
        if self.__commence:
            if not self.__pause:
                self.__forme.a_droite()

def forme_tourne(self):
        '''ModeleTetris -> None
        Fait tourner la forme à 90°
        '''
        if self.__commence:
            if not self.__pause:
                self.__forme.tourne()
```

Ces méthodes font appel à d'autre fonctions pour modifier le positionnement de la forme si le jeu n'est pas en pause pour éviter de pourvoir la bouger quand le jeu est arrêté.

```
def est_ligne_complete(self,lig):
        '''ModeleTetris -> Bool
        Verifie si une ligne du tableau est complète
        '''
        for i in range(len(self.__terrain[lig])):
            if not self.est_occupe(lig, i):
                return False
        return True
```

Cette méthode de la class ModeleTetris vérifie qu'une ligne précise du tableau est complète, si tout les éléments de la liste qui compose la ligne ne contient pas de -1.

```
def supprime_ligne(self,lig):
        '''ModeleTetris -> None
        Supprime la liste lig du terrain et ajoute une ligne
        '''
        del self.__terrain[lig]
        new_lig = [-1 for _ in range(self.__larg)]
        self.__terrain.insert(self.__base, new_lig)
```

Cette méthode va donc supprimer les lignes complètes et en ajoute une nouvelle.

```
def supprime_lignes_completes(self):
        '''ModeleTetris -> None
        Supprime toutes les lignes complètes et ajoute 1 au score
        '''
        for lig in range(self.__base, self.__haut):
            if self.est_ligne_complete(lig):
                self.supprime_ligne(lig)
                self.__score += 1
```

Cette méthode va faire appel aux deux dernières pour s'occuper de supprimer les lignes complétées.

```
def get_score(self):
        '''ModeleTetris -> int
        Retourne le score
        '''
        return self.__score
```

Permet de retourner le score pour y avoir accès depuis les autres class.

```
def get_coords_suivante(self):
        '''ModeleTetris -> tuple(Int, Int)
        Récupère les coordonnées de la forme suivante
        '''
        return self.__suivante.get_coords_relatives()

    def get_couleur_suivante(self):
        '''ModeleTetris -> Int
        Retourne la valeur/couleur de la forme suivante
        '''
        return self.__suivante.get_couleur()
```

Permettent de récupérer les informations qui sont les coordonnés relative et absolues ainsi que la couleur de la pièce en attente / la pièce suivante.

```
def get_commence(self):
        '''ModeleTetris -> bool
        Permet de savoir si le jeu a commencé ou pas
        '''
        return self.__commence

def get_pause(self):
        '''ModeleTetris -> bool
        Permet de savoir si le jeu est en pause ou pas
        '''
        return self.__pause
```

Permettent de récupérer les informations lié au jeu : si il est en pause et si il est commencé.

```
def maj_jeu(self):
        '''ModeleTetris -> None
        Met à jour le bouton quand on clique dessus
        '''
        if self.__commence:
            if self.__pause:
                self.__pause = False
            else:
                self.__pause = True
        else:
            self.__commence = True
```

Cette méthode va mettre à jour les informations, décrites ci dessus, contenu dans l'élément de la class ModeleTetris quand on clique sur le bouton correspondant.

```
def reco(self):
        self.__score = 0
        self.__terrain:list = [[-2 for _ in range(self.__larg)] for _ in range(self.__base)] + [[-1 for _ in range(self.__larg)] for _
        self.__forme = Forme(self)
        self.__suivante = Forme(self)
        self.__commence = False
        self.__pause = False
```

Permet de recommencer la partie en recréant un élément de la class ModeleTetris pour remplacer l'ancien.

```
class Forme:
    def __init__ (self, modele):
        self.__modele = modele
        indice = randint(0,6)
        self.__couleur = indice
        self.__forme = LES_FORMES[indice]
        self.__x0 = randint(1,self.__modele.get_largeur()-2)
        self.__y0 = 1
```

Initialisation de la classe Forme qui va récupérer les informations de la classe ModeleTetris dans self.__modele. La forme est créer avec des coordonnées qui vont se baser sur son centre qui à les coordonnées (0,0). self.__x0 et self.__y0 vont servir à définir les coordonnés de la forme. Le x qui correspond à la colonne où se positionne la forme et donc choisit aléatoirement.

```
def get_couleur(self):
        '''Forme -> int
        Récupère la couleur en cours
        '''
        return self.__couleur
```

Cette méthode sert à connaitre la couleur d'une case donc la valeur associé à partir de n'importe quel fichier.

```
def get_coords(self):
        '''Forme -> list(tuple(int))
        Récupère les coordonnées absolue de la forme
        '''
        res = []
        for elmt in self.__forme:
            res.append((elmt[0] + self.__x0, elmt[1] + self.__y0))
        return res
```

La méthode get_coords() va être utiliser pour placer la forme puis pour la faire descendre en fonction de l'avancé du jeu.

```
def collision(self):
        '''Forme,Forme-> Bool
        Vérifie si la forme touche une autre pièce ou le bas du terrain
        '''
        for co in self.get_coords():
            if co[1] == self.__modele.get_hauteur()-1:
                return True
        for elem in self.get_coords():
            x,y = elem
            if self.__modele.est_occupe(y+1,x):
                return True
        return False
```

Cette méthode va permettre de vérifier que la forme peut continuer de tomber, tant qu'elle n'a pas touché le bas du terrain avec la première partie du code. La deuxième partie vérifie que la case suivante, n'est pas occupée et si c'est la cas, True est retourné.

```
def tombe(self):
        '''Forme -> Bool
        Fait tomber la forme
        '''
        if not self.collision():
            self.__y0 +=1
            return False
        return True
```

La méthode tombe() est donc lié avec celle vu juste avant car si il n'y pas de possibilité de collision, la pièce peut alors tomber en ajoutant 1 au y qui correspond à la ligne où se situe la forme.

```
def position_valide(self):
        '''Forme -> Bool
        Vérifie si une position est possible pour chaque coordonnées de la forme
        '''
        for elmt in self.get_coords():
            if self.__modele.est_occupe(elmt[1],elmt[0]):
                return False
            if elmt[0] < 0 or elmt[0] >= self.__modele.get_largeur():
                return False
            if not elmt[1] < self.__modele.get_hauteur()-1:
                return False
        return True
```

Cette méthode à pour but de regarder toutes les coordonnées qui composent une forme et vérifie que chacune d'entre elle peut exister.

```
def a_gauche(self):
        '''Forme -> None
        Fait se diriger la forme vers la gauche
        '''
        self.__x0 -= 1
        if not self.position_valide():
            self.__x0 +=1

def a_droite(self):
        '''Forme -> None
        Fait se diriger la forme vers la droite
        '''
        self.__x0 += 1
        if not self.position_valide():
            self.__x0 -=1
```

Ces deux méthodes vont modifier les coordonnées qui compose une forme en y ajoutant 1 ou -1 en fonction de la direction où elle doit se diriger. Bien sûr, si la position n'est pas valide, les coordonnées reprennent 1 ou -1 pour annuler le changement précédent.

```
def tourne(self):
        '''Forme -> None
        Fait tourner la forme à 90°
        '''
        forme_prec = self.__forme
        self.__forme=[]
        for x,y in forme_prec:
            self.__forme.append((-y, x))
        if not self.position_valide():
            self.__forme = forme_prec
```

La méthode tourne() permet de faire tourner la pièce de 90° vers la droite. Elle est d'abord enregistrée afin d'annuler les changements si la position n'est pas valide. Après cela, la pièce de base subit une rotation à l'aide de forme_prec et (-y, x).

```
def get_coords_relatives(self):
        '''Forme -> list(tuple(int))
        Récupère les coordonnées relatives de la forme
        '''
        return self.__forme.copy()
```

Permet de récupérer les coordonnés relative de la pièce en attente / la pièce suivante.

Fichier vue.py

```
class VueTetris:
    def __init__(self, modele_tetris):
        self.__modele = modele_tetris
        self.__fenetre = tk.Tk()
        self.__fenetre.title("Tetris")

        self.__can_terrain = tk.Canvas(self.__fenetre, width=self.__modele.get_largeur() * DIM, height=self.__modele.get_hauteur() * D
        self.__can_terrain.pack(side="left")
        self.__cases = []
        for i in range(self.__modele.get_hauteur()):
            ligne_cases = []
            for j in range(self.__modele.get_largeur()):
                case = self.__can_terrain.create_rectangle(j * DIM, i * DIM, (j + 1) * DIM, (i + 1) * DIM, outline="grey", fill="")
                ligne_cases.append(case)
            self.__cases.append(ligne_cases)

        self.__boutons = tk.Frame(self.__fenetre)


        self.__jeu = tk.Button(self.__boutons, text="", command=self.__modele.maj_jeu)
        self.__jeu.pack()

        tk.Label(self.__boutons,text="Forme suivante").pack()
        self.__can_fsuivante = tk.Canvas(self.__boutons, width=SUIVANT* DIM, height=SUIVANT*DIM )
        self.__can_fsuivante.pack()


        self.__lbl_score = "Score : 0"
        self.__score = tk.Label(self.__boutons, text = self.__lbl_score, fg="red")
        self.__score.pack()
        self.__les_suivants=[]
        for i in range(SUIVANT):
            ligne_cases_suiv=[]
            for j in range (SUIVANT):
                case_suiv = self.__can_fsuivante.create_rectangle(j*DIM, i*DIM, (j+1)*DIM, (i+1)*DIM, outline="grey", fill ="black")
                ligne_cases_suiv.append(case_suiv)
            self.__les_suivants.append(ligne_cases_suiv)

        self.__quitter = tk.Button(self.__boutons, text="Quitter", command=self.__fenetre.destroy)
        self.__quitter.pack()

        self.__touches1 = tk.Label(self.__boutons, text = "Flèches Gauche/Droite = Diriger", fg="green")
        self.__touches2 = tk.Label(self.__boutons, text = "Flèche Bas = Tomber plus vite", fg="green")
        self.__touches3 = tk.Label(self.__boutons, text = "Barre espace = Tomber encore plus vite", fg="green")
        self.__touches4 = tk.Label(self.__boutons, text = "Flèche Haut = Tourner Forme", fg="green")
        self.__touches1.pack()
        self.__touches2.pack()
        self.__touches3.pack()
        self.__touches4.pack()


        self.__boutons.pack(side="right")
```

Initialisation de la classe VueTetris qui correspond à la partie graphique du jeu. La classe créer donc la page avec comme titre Tetris puis s’occupe du canvas qui va accueillir le terrain avec toutes les cases qui sont créées avec les lignes situées juste en dessous. La liste self.__les_cases va servir à stocker toute les case qui composent le terrain afin de les modifier au fur et à mesure de l’avancé du jeu.

```
def get_fenetre(self):
        '''VueTetris -> VueTetris.__fenetre
        Récupère la fenêtre Tetris
        '''
        return self.__fenetre
```

Cette méthode retourne simplement l’instance de Tk de l’application.

```
def dessine_case(self, j, i, coul):
        '''VueTetris, Int, Int, Int -> None
        Change la couleur de la case
        '''
        self.__can_terrain.itemconfigure(self.__cases[i][j], fill=COULEURS[coul])
```

La méthode dessine_case() va s'occuper de modifier les cases stockées dans la liste self.__les_cases comme dit plus tôt avec une couleur défini dans les paramètres qui va être leur valeur

```
def dessine_terrain(self):
        '''VueTetris -> None
        Dessine chaque case du terrain
        '''
        for i in range(self.__modele.get_hauteur()):
            for j in range(self.__modele.get_largeur()):
                coul = self.__modele.get_terrain()[i][j]
                self.dessine_case(j, i, coul)
```

Cette méthode sert à dessiner donc le terrain sans les formes en récupérant les valeurs de chaque case que contient le terrain afin de choisir leur couleur plus tard dans la méthode dessine_case().

```
def dessine_forme(self, coords, couleur):
        '''VueTetris, tuples(Int, Int), Int -> None
        Dessine chaque case de la forme
        '''
        for x, y in coords:
            self.dessine_case(x, y, couleur)
```

C'est cette méthode qui va permettre d'afficher les formes en utilisant la même méthode dessine_case(). L'utilisation de dessine_terrain() va permettre d'effacer l'ancien emplacment de la forme et l'utilisation de dessine_forme() permet donc de la dessiner à son nouvel emplacement.

```
def met_a_jour_score(self, val):
        '''VueTetris, Int-> None
        Change la valeur du score
        '''
        self.__score.configure(text="Score : "+ str(val))
```

Permet de mettre à jour l'affichage du score sur la fenêtre Tetris.

```
def dessine_case_suivante(self, x, y, coul):
        '''VueTetris, Int, Int, Int -> None
        Change la couleur de la case
        '''
        self.__can_fsuivante.itemconfigure(self.__les_suivants[x][y], fill=COULEURS[coul])


def nettoie_forme_suivante(self):
        '''VueTetris-> None
        Remet toute les cases de la partie pièce suivante en noir
        '''
        for i in range(len(self.__les_suivants)):
            for j in range(len(self.__les_suivants[i])):
                self.__can_fsuivante.itemconfigure(self.__les_suivants[i][j], fill="black")


def dessine_forme_suivante(self, coords, coul):
        '''VueTetris, tuples(Int, Int), Int -> None
        Dessine chaque case de la forme suivante
        '''
        self.nettoie_forme_suivante()
        for x, y in coords:
            self.dessine_case_suivante(y+2, x+2, coul)
```

Ces méthodes vont s'occuper de l'affichage de la forme suivante en dessinant les cases qui la composent et en nettoyant la carré où la forme est dessinée afin d'afficher la suivante.

```
def met_a_jour_bouton(self):
        '''VueTetris -> None
        Permet de changer le texte dans le bouton qui lance et met le jeu en pause
        '''
        if not self.__modele.get_commence():
            self.__jeu.configure(text="Commencer")
        elif not self.__modele.get_pause():
            self.__jeu.configure(text="Pause")
        elif self.__modele.get_pause():
            self.__jeu.configure(text="Reprendre")

    def met_a_jour_reco(self):
        '''VueTetris -> None
        Permet de changer le texte dans le bouton pour relancer le jeu
        '''
        self.__jeu.configure(text="Recommencer", command = self.__modele.reco)

    def met_a_jour_co(self):
        '''VueTetris -> None
        Permet de changer le texte dans le bouton pour relancer le jeu
        '''
        self.__jeu.configure(text="Recommencer", command = self.__modele.maj_jeu)
```

Ces méthodes permettent de mettre à jour les différentes valeurs qui composent le bouton tel que le texte qui est affiché et la fonction qui va être effectué quand on clique dessus.

Fichier pytetris.py

```
class Controleur:
    def __init__(self, tetris):
        self.__tetris:modele.ModeleTetris = tetris
        self.__vue = vue.VueTetris(self.__tetris)
        self.__fen = self.__vue.get_fenetre()
        self.__fen.bind("<Key-Left>",self.forme_a_gauche)
        self.__fen.bind("<Key-Right>",self.forme_a_droite)
        self.__fen.bind("<Key-Down>",self.forme_tombe)
        self.__fen.bind("<Key-Up>",self.forme_tourne)
        self.__fen.bind("<space>",self.forme_tombe_a_fond)
        self.__delai = 260
        self.__delai2 = 0
        self.joue()
        self.__fen.mainloop()
```

Initialisation de la classe Controleur prend une instance du modèle en paramètre et il le conserve dans un attribut. Puis il créé une instance de la VueTetris, auquel il transmet le modele et le conserve également dans un attribut. Les lignes “*self.__fen.bind("<…>",self.forme_…)*” vont servir attribuer des touches du clavier à des fonctions.

```
def joue(self) :
        '''Controleur -> None
        boucle principale du jeu. Fait tomber une forme d’une ligne.
        '''
        self.__vue.met_a_jour_co()
        if not self.__tetris.fini() :
            self.__vue.met_a_jour_bouton()
            self.affichage()
            self.__fen.after(self.__delai,self.joue)

        else:
            self.__vue.met_a_jour_reco()
            self.__delai = 260
            self.__fen.after(self.__delai,self.joue)
```

Cette méthode est la boucle principal du jeu et teste si la partie est finie, puis appelle la méthode affichage de la classe Controleur. Ensuite, la méthode *after* de Tk permet de rappeler la méthode joue() au bout de self.__delai millisecondes d’attente. Quand la partie est finie, on a donc la possibilité de relancer avec le bouton et la partie *else* de ce code.

```
def affichage(self):
        '''Controleur -> None
        Dessine le terrain et la forme si elle est entrain de tomber
        '''
        self.__vue.dessine_terrain()
        self.__vue.dessine_forme(self.__tetris.get_coords_forme(), self.__tetris.get_couleur_forme())
        self.__vue.dessine_forme_suivante(self.__tetris.get_coords_suivante(),self.__tetris.get_couleur_suivante())
        if self.__tetris.get_commence():
            if not self.__tetris.get_pause():
                if not self.__tetris.forme_tombe():
                    self.__vue.dessine_terrain()
                    self.__vue.dessine_forme(self.__tetris.get_coords_forme(), self.__tetris.get_couleur_forme())
                else:
                    self.tombe_paliers()
                    val = self.__tetris.get_score()
                    self.__vue.met_a_jour_score(val)
                    self.__vue.dessine_forme_suivante(self.__tetris.get_coords_suivante(),self.__tetris.get_couleur_suivante())
```

La méthode affichage() va donc faire appelle aux deux méthodes vu précédemment pour dessiner le terrain et la forme si elle est entrain de tomber. Par la suite, on vérifie bien que le jeu n'est pas en pause et à bien commencé avant de modifié les couleurs du terrain.

```
def forme_a_gauche(self, event):
        '''Controleur -> None
        Fait se diriger la forme vers la gauche
        '''
        self.__tetris.forme_a_gauche()

def forme_a_droite(self,event):
        '''Controleur -> None
        Fait se diriger la forme vers la droite'''
        self.__tetris.forme_a_droite()

def forme_tombe(self,event):
        '''Controleur -> None
        Fait tomber la forme plus vite
        '''
        if self.__tetris.get_commence():
            if not self.__tetris.get_pause():
                self.__delai = 100

def forme_tombe_a_fond(self,event):
        '''Controleur -> None
        Fait tomber la forme plus vite
        '''
        if self.__tetris.get_commence():
            if not self.__tetris.get_pause():
                self.__delai = 1

def forme_tourne(self, event):
        '''Controleur -> None
        Fait tourner la forme à 90°
        '''
        self.__tetris.forme_tourne()
```

Ces cinq méthodes vont servir à appeler les fonctions qui font bouger la forme comme le joueur le souhaite. Cela grâce aux cinq lignes ajoutées au constructeur qui vont faire appel à ces méthodes quand les touches du clavier associées seront utilisées.

```
def tombe_paliers(self):
        '''Controleur -> None
        Fait tomber la pièce de plus en plus vite en fonction du score
        '''
        self.__delai = 260
        if self.__tetris.get_score() < 10 :
            self.__delai = 260 - self.__tetris.get_score()*10
            self.__delai2 = self.__delai
        else:
            self.__delai = self.__delai2
```

Cette méthode permet d'accéler le jeu au fur et à mesure que le score augmente en diminuant *self.__delai*. Afin de ne pas rendre le jeu impossible, si la score est de 10, le jeu ne pas pas aller plus vite.

```
if __name__ == "__main__" :
    # création du modèle
    tet = modele.ModeleTetris()
    # création du contrôleur. c’est lui qui créé la vue
    # et lance la boucle d’écoute des évts
    ctrl = Controleur(tet)
```

Programme principale qui permet de lancer le jeu.