Le Projet: Tetris

Fouet Quentin - Brun Dylan

Modele.py

ModeleTetris

```
LES_FORMES:list = [[(0,0),(0,-1),(0,1),(0,2)], [(0,0),(0,1),(1,1),(1,0)], [(0,0),(1,0),(0,1),(-1,1)], [(0,0),
(-1,0),(0,1),(1,1)], [(-1,1),(-1,0),(0,0),(1,0)], [(0,0),(-1,0),(1,0),(1,1)], [(0,0),(-1,0),(1,0),(0,-1)]]

class ModeleTetris:
    #Constructeur de la classe ModeleTetris.
    def __init__(self, lignes:int=18, colonnes:int=13):
        self.__base:int = 4
        self.__score:int = 0
        self.__haut:int = lignes + self.__base
        self.__larg:int = colonnes
        self.__terrain:list = [[-1 for _ in range(self.__larg)] for _ in range(self.__haut)]
        for i in range(4):
            self.__terrain[i] = [-2 for _ in range(self.__larg)]
        self.__forme:Forme = Forme(self)
        self.__suivante:Forme = Forme(self)
        return

    #Retourne la hauteur du terrain.
    def get_hauteur(self):
        return self.__haut

    #Retourne la largeur du terrain.
    def get_largeur(self):
        return self.__larg

    #Retourne la valeur qu'il y a à aux coordonnées colonne ligne.
    def get_valeur(self, ligne, colonne):
        return self.__terrain[ligne][colonne]

    #Vérifie si l'emplacement aux coordonnées colonne ligne est occupé. (c'est-à-dire que la valeur est plus
grande ou égal à 0)
    def est_occupe(self, ligne, colonne):
        if ligne >= self.__haut or ligne < 0 or colonne >= self.__larg or colonne < 0:
            return True
        return self.__terrain[ligne][colonne] >= 0

    #Vérifie si la partie est fini ou pas.
    def fini(self):
        for colonne in range(self.__larg):
            if self.__terrain[self.__base-1][colonne] != -2:
                return True
        return False

    #Ajoute la forme au terrain.
    def ajoute_forme(self):
        for y, x in self.__forme.get_coords():
            self.__terrain[y][x] = self.__forme.get_couleur()
```

```
        return

    #Essaie de faire tomber la forme, si elle n'y arrive pas, alors l'ancienne forme sera ajoutée au terrain et
une nouvelle sera créée.
    def forme_tombe(self):
        if self.__forme.tombe():
            self.ajoute_forme()
            self.supprimes_lignes_completes()
            self.__forme = self.__suivante
            self.__suivante = Forme(self)
            return True
        return False

    #Retourne la couleur de la forme.
    def get_couleur_forme(self):
        return self.__forme.get_couleur()

    #Retourne les coordonnées de la forme.
    def get_coords_forme(self):
        return self.__forme.get_coords()

    #Fait déplacer la forme à gauche.
    def forme_a_gauche(self):
        self.__forme.a_gauche()
        return

    #Fait déplacer la forme à droite.
    def forme_a_droite(self):
        self.__forme.a_droite()
        return

    #Fait une rotation de la forme.
    def forme_tourne(self):
        self.__forme.tourne()
        return

    #Teste si la ligne lig du terrain est complète.
    def est_ligne_complete(self, lig:int):
        for elem in self.__terrain[lig]:
            if elem < 0:
                return False
        return True

    #Supprime la ligne d'indice lig sur le terrain, toutes les valeurs des lignes de self.__base à lig-1 inclus
descendent d'un cran.
    def supprime_ligne(self, lig:int):
        nouvelleL:list = [[-2 for _ in range(self.__larg)] for _ in range(self.__base)]
        nouvelleL.append([-1 for _ in range(self.__larg)])
        for i in range(4, lig):
            nouvelleL.append(self.__terrain[i])
        for j in range(lig+1, self.__haut):
            nouvelleL.append(self.__terrain[j])
        self.__terrain = deepcopy(nouvelleL)
        return

    #Supprime toutes les lignes complètes, et pour chaque lignes complètes supprimées, augmente le score de 1.
    def supprimes_lignes_completes(self):
        for i in range(4, self.__haut):
            if self.est_ligne_complete(i):
                self.supprime_ligne(i)
                self.__score += 1
        return

    #Retourne la valeur du score.
    def get_score(self):
        return self.__score

    #Retourne les coordonnées relatives de __suivante.
    def get_coords_suivante(self):
        return self.__suivante.get_coords_relative()

    #Retourne la couleur de __suivante.
```

```
    def get_couleur_suivante(self):
        return self.__suivante.get_couleur()

    #Réinitialise le terrain et ses formes.
    def reinitialise(self):
        self.__terrain:list = [[-1 for _ in range(self.__larg)] for _ in range(self.__haut)]
        for i in range(4):
            self.__terrain[i] = [-2 for _ in range(self.__larg)]
        self.__forme:Forme = Forme(self)
        self.__suivante:Forme = Forme(self)
        return
```

Forme

```
class Forme:
    #Constructeur de la classe Forme.
    def __init__(self, modele):
        self.__modele:ModeleTetris = modele
        self.__couleur:int = random.randint(0, len(LES_FORMES)-1)
        self.__forme:list = LES_FORMES[self.__couleur]
        self.__y0:int = 2
        self.__x0:int = random.randint(2,self.__modele.get_largeur()-2)
        return

    #Retourne la couleur de la forme.
    def get_couleur(self):
        return self.__couleur

    #Retourne les coordonnées de la forme.
    def get_coords(self):
        coords = [(self.__y0 + y, self.__x0 + x) for x, y in self.__forme]
        return coords

    #Teste s'il y a une collision en dessous de la forme.
    def collision(self):
        x:int = 0
        y:int = 0
        for i, j in self.__forme:
            x = self.__x0 + i
            y = self.__y0 + j
            if self.__modele.est_occupe(y+1, x):
                return True
        return False

    #Fais tomber la forme s'il n'y a pas de collision en dessous de celle-ci, retourne True si elle ne peut plus
tomber, False sinon.
    def tombe(self):
        if not self.collision():
            self.__y0 += 1
            return False
        else:
            return True

    #Vérifie si la prochaine position de la forme est valide.
    def position_valide(self, coords:tuple):
        y, x = coords
        return not self.__modele.est_occupe(y, x)

    #Vérifie si la prochaine position à gauche de la forme est valide.
    def a_gauche(self):
        coords:list = self.get_coords()
        for y,x in coords:
            if not self.position_valide((y, x-1)):
                return
        self.__x0 -= 1
        return
```

```
    #Vérifie si la prochaine position à droite de la forme est valide.
    def a_droite(self):
        coords:list = self.get_coords()
        for y,x in coords:
            if not self.position_valide((y, x+1)):
                return
        self.__x0 += 1
        return

    #Fait tourner la forme, si la position n'est pas valide, remet la forme à l'orientation d'avant.
    def tourne(self):
        forme_prec:list = self.__forme.copy()
        self.__forme = [(-y, x) for x, y in self.__forme]
        coords:list = self.get_coords()
        for coord in coords:
            if not self.position_valide(coord):
                self.__forme = forme_prec
                break
        return

    #Retourne une copie de la liste des coordonnées relatives de la forme.
    def get_coords_relative(self):
        return deepcopy(self.__forme)
```

PyTetris.py

Controleur

```
class Controleur:
    #Constructeur de la classe Controleur.
    def __init__(self, tetris):
        self.__tetris:modele.ModeleTetris = tetris
        self.__vue:vue.VueTetris = vue.VueTetris(tetris)
        self.__fen:vue.tk.Tk = self.__vue.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_instant)
        self.__fen.bind("<Escape>", self.pause_event)
        self.__delai:int = 210
        self.__delai_backup:int = self.__delai
        self.__valeur_choix:int = 0
        self.pause_:int = 0
        self.__manager:dict = {0:("Commencer", self.lance_partie), 1:("Pause", self.pause), 2:("Reprendre",
self.reprendre), 3:("Recommencer", self.recommencer)}
        self.change_bouton_mode()
        pygame.mixer.init()
        self.__fen.mainloop()
        return

    #Commence et fini la partie de Tetris.
    def joue(self):
        if not self.__tetris.fini():
            if not self.pause_:
                self.affichage()
                self.__vue.fenetre().after(self.__delai, func=self.joue)
            else:
                return
        else:
            self.__valeur_choix = 3
            self.change_bouton_mode()
            self.son_stop()
        return

    #Affiche la partie de Tetris.
    def affichage(self):
```

```
            if self.__tetris.forme_tombe():
                self.__delai = self.__delai_backup
            else:
                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())
                self.__vue.met_a_jour_score(self.__tetris.get_score())
            return

        #Fait aller la forme à gauche.
        def forme_a_gauche(self, event):
            self.__tetris.forme_a_gauche()
            return

        #Fait déplacer la forme à droite.
        def forme_a_droite(self, event):
            self.__tetris.forme_a_droite()
            return

        #Fait déplacer la forme plus rapidement.
        def forme_tombe(self, event):
            self.__delai = 10
            return

        #Fait une rotation de la forme.
        def forme_tourne(self, event):
            self.__tetris.forme_tourne()
            return

        #Modifie le bouton qui fait tout.
        def change_bouton_mode(self):
            text_, command_ = self.__manager[self.__valeur_choix]
            self.__vue.get_boutonquifaittout().configure(text=text_, command=command_)
            return

        #Lance la partie.
        def lance_partie(self):
            self.__valeur_choix += 1
            self.change_bouton_mode()
            self.joue()
            self.son_play()
            return

        #Met en pause la partie.
        def pause(self):
            self.__valeur_choix = 2
            self.change_bouton_mode()
            self.pause_ = 1
            self.__fen.unbind("<Key-Left>")
            self.__fen.unbind("<Key-Right>")
            self.__fen.unbind("<Key-Down>")
            self.__fen.unbind("<Key-Up>")
            self.__fen.unbind("<space>")
            self.__fen.bind("<Escape>", self.reprendre_event)
            self.son_pause()
            return

        #Met sur pause le jeu via le bouton Echap.
         def pause_event(self, event):
            if self.__valeur_choix == 1:
                self.pause()
            return

        #Reprend la partie.
        def reprendre(self):
            self.__valeur_choix = 1
            self.change_bouton_mode()
            self.pause_ = 0
            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_instant)
        self.__fen.bind("<Escape>", self.pause_event)
        self.joue()
        self.son_unpause()
        return

    #Reprend la partie via le bouton Echap.
    def reprendre_event(self, event):
        if self.__valeur_choix == 2:
            self.reprendre()
        return

    #Recommence la partie.
    def recommencer(self):
        self.__valeur_choix = 1
        self.change_bouton_mode()
        self.__delai = self.__delai_backup
        self.__tetris.reinitialise()
        self.__vue.met_a_jour_score(0)
        self.joue()
        self.son_play()
        return

    #Fais tomber la pièce instantanément.
    def forme_tombe_instant(self, event):
        self.__delai = 0
        return

    #Lance la musique du Tetris.
    def son_play(self):
        pygame.mixer.music.load("./sounds/tetris_background_soundtrack.mp3")
        pygame.mixer.music.play(loops=-1)
        return

    #Met en pause la musique du Tetris.
    def son_pause(self):
        pygame.mixer.music.pause()
        return

    #Reprend la musique du Tetris.
    def son_unpause(self):
        pygame.mixer.music.unpause()
        return

    #Stop la musique du Tetris.
    def son_stop(self):
        pygame.mixer.music.stop()
        return
```

Vue.py

VueTetris

```
DIM:int = 30
COULEURS:list = ["pink", "purple", "green", "blue", "red", "yellow","orange","dark grey","black"]
SUIVANT:int = 6

class VueTetris:
    #Constructeur de la classe VueTetris.
    def __init__(self, ModeleTetris:modele.ModeleTetris):
        self.__modele:modele.ModeleTetris = ModeleTetris
        self.__fenetre:tk.Tk = tk.Tk()
        fenbtn:tk.Frame = tk.Frame(self.__fenetre)
        self.__can_terrain = tk.Canvas(self.__fenetre, width=self.__modele.get_largeur() * DIM,
height=self.__modele.get_hauteur() * DIM, bg="grey")
```

```
        self.__can_fsuivante:tk.Canvas = tk.Canvas(fenbtn, width=SUIVANT * DIM, height=SUIVANT * DIM)
        self.__les_suivants:list = [[self.__can_fsuivante.create_rectangle(x*DIM, y*DIM, (x+1)*DIM, (y+1)*DIM,
outline="grey", fill="black") for x in range(SUIVANT)] for y in range(SUIVANT)]
        self.__lbl_score:tk.Label = tk.Label(fenbtn, text="Score : 0")
        self.boutonQuitter = tk.Button(fenbtn, text="Quitter", command=self.stop_music_and_close_window)
        self.__fenetre.protocol("WM_DELETE_WINDOW", self.stop_music_and_close_window)
        self.boutonquifaittout:tk.Button = tk.Button(fenbtn, text="")

        tk.Label(fenbtn, text="Forme suivante :").pack()
        self.__can_fsuivante.pack()
        self.__lbl_score.pack()
        self.boutonquifaittout.pack()
        self.boutonQuitter.pack()

        self.__can_terrain.pack(side="left")
        fenbtn.pack(side="right")

        self.__les_cases = []
        for y in range(self.__modele.get_hauteur()):
            ligne = []
            for x in range(self.__modele.get_largeur()):
                case = self.__can_terrain.create_rectangle(x*DIM, y*DIM, (x+1)*DIM, (y+1)*DIM, outline="grey",
fill=COULEURS[self.__modele.get_valeur(y,x)])
                ligne.append(case)
            self.__les_cases.append(ligne)
        return

    #Retourne la fenêtre.
    def fenetre(self):
        return self.__fenetre

    #Dessine une case de couleur COULEURS[coul] aux coordonnées x y.
    def dessine_case(self, y, x, coul):
        self.__can_terrain.itemconfigure(self.__les_cases[y][x], fill=COULEURS[coul])
        return

    #Dessine le terrain.
    def dessine_terrain(self):
        for y in range(self.__modele.get_hauteur()):
            for x in range(self.__modele.get_largeur()):
                self.dessine_case(y, x, self.__modele.get_valeur(y,x))
        return

    #Dessine la forme au coordonnées coords avec pour couleur COULEURS[couleur].
    def dessine_forme(self, coords, couleur):
        for coord in coords:
            y, x = coord
            self.dessine_case(y, x, couleur)
        return

    #Change le texte de __lbl_score pour afficher val dans le score.
    def met_a_jour_score(self, val:int):
        self.__lbl_score.configure(text=f"Score : {val}")
        return

    #Dessine une case de couleur COULEURS[coul] aux coordonnées x y dans le canvas __can_fsuivante.
    def dessine_case_suivante(self, x:int, y:int, coul:int):
        self.__can_fsuivante.itemconfigure(self.__les_suivants[y][x], fill=COULEURS[coul])
        return

    #Remet du noir sur tous les carrés de __can_fsuivante.
    def nettoie_forme_suivante(self):
        for y in range(SUIVANT):
            for x in range(SUIVANT):
                self.dessine_case_suivante(x, y, -1)
        return

    #Nettoie le Canvas fsuivante, puis dessine la forme suivante dedans.
    def dessine_forme_suivante(self, coords:list, coul:int):
        self.nettoie_forme_suivante()
        for x,y in coords:
            self.dessine_case_suivante(x+2, y+2, coul)
```

```
            return

        #Retourne le bouton qui fait tout.
        def get_boutonquifaittout(self):
            return self.boutonquifaittout

        #Ferme la fenêtre et stop la musique en même temps.
        def stop_music_and_close_window(self):
            pygame.mixer.music.stop()
            self.__fenetre.destroy()
```