SAE (ALGO2) : Projet Tetris

Projet réalisé en Binôme : 2022/2023

1- **MACHTER Massinissa grp : TD5**

2- **KHAZEM Lynda grp : TD5**

Un descriptif des différentes méthodes utilisées dans notre Application *:*

Etape1 :

Dans cette étape on a réalisé une version du jeu ou une forme tombe toute seule sur le terrain. Lorsqu'elle se pose, une nouvelle forme se met à tomber toute seule, tandis que le jeu s'arrête lorsque le terrain est rempli.

NB **:** Le joueur ne peut pas intervenir dans cette étape que pour quitter l'app.

Ensemble des classes et fonctions crées :

Le modèle :

Class ModeleTetris (c'est le modèle de jeu , le moteur de jeu) :

Dans le constructeur du modèle on a modifie certains attributs comme :

1/ 5 lignes sont ajouté (au lieu de 4) au lanceur de formes «la zone grisé » (cela pour bien afficher les formes qu'on a ajouté)

2/ self.__base on l'a mis a 5 au lieu de 4 (cela pour bien afficher les formes qu'on a ajouté)

On a crée les fonctions que vous avez demandé dans vos instructions tels que (get_largeur() ,get_hauteur() , etc…

Class Forme (la class forme prend une instance du modèle tetris) :

la class Forme c'est celle qui prend en charges tous ce qui est formes dans le jeu , elle est reliée a la classe modèle car elle prend une instance de "ModeleTetris" en paramètre.

Dans cette pour la class formes on n'a pas vraiment modifie les fonctions demandé et le constructeur

Seulement on indique que dans les coords relatives et absolues on a utilisé le terme « tup » pour parcourir les tuples dans la liste des coords (list(tup)) , on tient a préciser que dans tous l'app tup[0] représente le x(c-a-d il parcourt l'axe principale), tup[0] représente le y (c-a-d il parcourt l'axe secondaire).

Sinon pour le reste des fonctions(collision ,tombe…) on les a crée en suivant les indications données dans l'étape 1 .

Class Vue (interface graphique – interactions avec l'utilisateur) :

Cette class modélise tous ce qui est affichage et graphisme, la fenêtre de jeu c.-à-d. l'interface et le contact avec l'utilisateur.

le constructeur de la classe de VueTetris prend en paramètre une instance de la classe ModeleTetris ,

Elle construit le Windows principale (la fenêtre) de l'application toutes ses composantes.

Class Contrôleur (Lien entre Le modèle et la vue de jeu) :

Son constructeur prend en paramètre une instance de Modèle de jeu et il crée une instance de la vue (VueTetris) a travers du modèle passé en paramètre.

On a crée les méthodes (affichage, joue) et on a implémenté le script principal qui permet de jouer en bas du fichier pytetris.py

Etape 2:

NB **:** Maintenant le joueur peut agir dans cette étape sur la forme, même joué avec plusieurs types de forme.

Lorsque l'utilisateur appuie sur une touche (la flèche gauche ou la flèche droite) pour déplacer la forme latéralement événement soit écoutée par un contrôleur et parvenir le modèle qu'il doit bouger latéralement la forme si c'est possible, pour cela on a créé les fonctions :

Dans la class Forme :

Position_valide() -> qui teste si chaque coordonnée absolue de la forme est valide.

a gauche et **a_droite** -> qui déplace la forme d'une colonne vers la gauche et la droite .

Dans la classe ModeleTetris :

On a ajouté les deux méthodes **forme_a_gauche(et a_droite)** qui demande a la forme de tourner a

Gauche (a droite).

Dans la classe Contrôleur :

On a ajouté les deux méthodes **forme_a_gauche (self,event)** et **forme_a_droite(self,event**) qui prend en paramètre « **event** » pour faire relier les touches (clique droit et clique gauche) au déplacement latérale , cela se fait en demandent a la forme de tourner a gauche(ou a droite).

Faire tomber plus vite une forme

NB **:** Maintenant le joueur peut agir dans cette étape sur la vitesse de jeu (faire tomber vite la forme), pour cela on a met quelques modification au niveau du contrôleur :

- Création d'une méthode **forme_tombe(self,event)** qui modifie la valeur de l'attribut **self.__delai**(on le diminue) c.-à-d. la forme s'accélère en descendant .
- Liez l'événement appui sur la flèche bas (Key-Down) a la méthode **forme_tombe**

- Légère modification au niveau de la méthode affichage, pour que a l'appui sur la flèche bas doit faite tomber plus vite une forme, et la vitesse normale doit être rétablie a la forme suivante (remettre l'attribut self .__delai a sa valeur initiale).

Tourner une forme :

Faire tourner une forme, veut dire qu'on change les coordonnées relatives qui d'écrivent la forme, pour cela nous faisons des rotations de 90 ˚ par rapport a l'origine de notre repère « notre pivot (x0,y0) »

On a implémenté :

- La **methode tourne** dans la class Forme qui fait la rotation de 90 deg si c'est c possible (change les coords relatives), C.-à-d. elle fait tourner la forme si c'est possible (si la position est valide « utilisation de la méthode **position_valide()** »
- Implémentation la méthode forme tourne de la classe **ModeleTetris**, elle demande a la forme de **tourner** .
- Implémentation la méthode forme **tourne(self,event)** de la classe Contrôleur qui demande au modèle de faire tourner la forme et on a liez cette méthode a l'événement « clique sur touche bas »

Plusieurs formes :

Dans le fichier modele.py, Nous avons initialisez une constante LES FORMES qui une liste des coords relatives des sept formes qu'on trouve habituellement dans le jeu comme une liste de liste de tuples (int,int).

NB : Dans le constructeur de la classe Forme, tirez un indice au hasard dans l'intervalle des indices possibles de la liste LES FORMES (0,len(LES_FORMES)). Cet indice deviendra la valeur de couleur dans la constante Couleurs dans le fichier vue.py

Rq : Dans les coords relatives des différentes formes, on a choisie le pivot (0,0) a la case la plus haute horizontalement possible de la forme, cela pour évitez tout affichage fugace (pour évitez que sur la ligne tout en bas du terrain que la forme déborde du cadre par le haut)

Etape3 :

Pour cette troisième étape, on s'intéresse aux : d'une part, lorsqu'une ligne est complète, elle sera supprimée, d'autre part, le jeu affiche la prochaine forme qui va tomber pour que l'utilisateur puisse anticiper ses actions et faire ses calcules.

Suppression des lignes pleines :

Il s'agit de supprimer du terrain les lignes pleines, en faisant tomber les lignes supérieures d'un cran. On a implémentez dans la classe **ModeleTetris** :

- On a ajouté un attribut **self .__score** qui va contenir le score de jeu .
- Implémentation la méthode **est_ligne_complete(self, lig)** qui teste si la ligne d'indice « lig » sur le terrain est complète(a la ligne lig on parcoure toutes les colonnes de cette lignes pour savoir si elle sont tous occupées)
- Implémentation la méthode **supprime_ligne(self,lig)** qui supprime la ligne d'indice lig sur le terrain et Toutes les valeurs des lignes de **self. __base** a **lig-1** « inclus » descendent d'un cran .
- Implémentation la méthode **supprime_lignes_completes(self)** qui supprime toutes les lignes complètes de jeu , a chaque ligne supprimée on augmente le score de 1.

Afficher le score(le nombre de lignes supprimées) :

L'idée est la suivante :

Dans la classe ModeleTetris, on a implémentez la méthode **get_score(self)** qui retourne la valeur du score, pour qu'on puisse le mettre a jour dans la méthode affichage() de contrôleur par la méthode implémentez « **met a jour score(self, val)** » de la class VueTetris en lui passant en paramètre self.__modele.get_score() .

Afficher la forme suivante :

On a fait apparaitre dans la zone a droite du terrain, la prochaine forme qui apparaitra en l'affichant sur un canvas (**self.__can_fsuivante**)similaire a celui qui permet de dessiner le terrain, mais d'une taille SUIVANT×SUIVANT carres de dimension.

NB : On a choisie de mettre **SUIVANT=7** au lieu de 6 pour bien afficher les formes qu'on a ajoutez.

Dans la classe VueTetris :

On a implémentez les méthodes : **dessine_case_suivante(x,y,coul)** qui dessine la case de la forme suivante au coords x,y et de couleur coul sur le canvas (self.__can_fsuivante), **nettoie_forme_suivante()** qui remet du noir sur tous les carres de self.__can_fsuivante pour pouvoir afficher une autre forme, **dessine_forme_suivante(coords,coul)** qui dessine la forme dont les coordonnées et la couleur sont données en paramètres , avant qu'elle dessine la forme suivante elle nettoie le Canvas et redessine les carres de coordonnées (x+3,y+3), et non pas (x,y) ses coords c'est le contrôleur qui va nous transmettre(c'est les coordonnées relatives de la forme).

NB : (x+3 ,y+3) et non pas (x+2,x+2) car nous avons choisie SUIVANT=7 donc on s'adapte avec .

Dans le Modèle :

- **Dans la classe Forme**, ajoutez une méthode g**et_coords()** relatives qui retourne une copie de la liste des coordonnées relatives de la forme.

- **Dans la classe ModeleTetris :** l'idée c'est qu'on a ajoutez un nouvel attribut self.__suivante et on la initialisez comme une nouvelle forme, implémentez une méthode **get_coords_ suivante ()** et **get_couleur_suivante()** qui retourne les coordonnées relatives de la forme suivante et sa couleur pour qu'on les passe en paramètre a la méthode de la vue **dessine_forme_suivante(coords,coul)**.

  Légère modification dans la méthode **forme_tombe** pour quand il y a eu collision et que nous avons ajouté la forme au terrain, alors la forme courante prend la valeur de la forme suivante, et la forme suivante est réinitialisée par une nouvelle Forme.

Dans le contrôleur :

Dans la méthode **affichage(),** si la forme courante s'est posée sur le terrain, on récupère auprès du modèle les coords relatives de la forme suivante ainsi que sa couleur, et qu'on passe a la vue pour qu'il affiche la forme suivante.

Etape 4 **(** cette dernière étape est dédiée a l'ajout de quelques dernières fonctionnalités et a personnaliser un peu le jeu **) :**

Faire une pause :

L'idée c'est qu'on a ajoutez un bouton qui permet de démarrer l'application lorsque le joueur clique dessus, ce bouton porte un texte différent au cours de la partie âpres le démarrage de la partie (pause et reprendre) et recommencer lorsque la partie est finie , alors :

Dans la Class VueTetris :

- On a créé dans le constructeur un nouvel attribut **self.__btn_crpr** qui est un composant tkinter (button) qu'on initialise a « Commencer » et qu'on relié comme command a la méthode **met_a_jour_btn(self).**(**NB :** crpr -> commencer, recommencer, pause, reprendre)
- on a créé et initialisé l'attribut **self.__etat** a False qui contient l'état de jeu (si la partie elle est en pause ou nn , si elle est commencer ou non , si elle redémarre ou nn).
- Implémentation de la méthode **change_etat(self)** qui change l'état de jeu c.-à-d. elle modifie le **self.__etat** (si il est a False elle le remet a True et vice-versa)
- Implémentation de la méthode **etat_de_jeu(self)** qui retourne l'état de jeu (retourne l'attribut self.__etat)
- Implémentation de la méthode **botton_crpr(self)** qui retourne self.__btn_crpr(le botton pour commencer, pause...) pour qu'on puisse agir sur lui dans le Contrôleur .
- Implémentation de la méthode **met_a_jour_btn(self)** ,elle mis a jour le composant (self.__btn_crpr),c'est-à-dire elle change le texte de ce Botton(pause ,reprendre ,recommencer) de telle sorte que si le texte de Botton était (commencer ou reprendre ou recommencer) et l'utilisateur tape dessus donc on le remet a pause et l'état de jeu a True, sinon on remet le texte de Botton a reprendre et l'état de jeu a False (car le joueur a cliqué sur pause). **NB :** elle change l'état de jeu a l'aide de la méthode change_etat(self)
- Implémentation de la méthode réinitialisation (self,mod) qui prend en paramètre un nouveau modèle(mod) est l'affecte a self.__modele

Dans le Contrôleur :

- Dans la méthode affichage (self) : on a ajouté une condition d'affichage de jeu selon son état (c.-à-d. on vérifie l'état de jeu avant de faire tomber ou de dessiner le terrain)
- Modification au niveau de la méthode joue (self) : si la partie est finie alors : on demande a la vue de changer l'état de jeu (change_etat()) c.-à-d. la partie est finie donc on est en pause, et on modifie le Botton pour afficher « Recommencer » ,on réinitialise le modèle a un nouveau modèle(on a déjà créé une méthode « **réinitialisation(self)** »dans la class modèle qui réinitialise seulement le modèle de jeu a une nouvelle instance ModeleTetris()) et on demande a la vue de réinitialisez son modèle a ce nouveau modèle retourner par la méthode **réinitialisation(self)** du modele.py et finalement on fait un appel récursif a la méthode joue(),c-a-d on attend du joueur de lancé une nouvelle partie .
- On a fait une liaison dans le constructeur entre la touche d'entrée <Return> et le Botton (self .__btn_crpr) pour commencer, faire une pause, reprendre et recommencer : pour cela on a implémentez dans la class Contrôleur la méthode **change_btn(self)** qui demande en occurrence a la vue de mettre a jour ce bouton .

Quelque autres améliorations(personnalisation de jeu) :

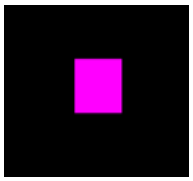
Afficher le message Game over lorsque la partie est finie:

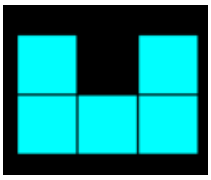
Pour cela on ajouté un composant (Label) dans le constructeur de la class VueTetris self.__btn_game a qui on initialise son texte a vide et on a implémentez les deux méthodes affiche et efface game_over(self) , pour que dans le Contrôleur lorsque la partie est finie on demande a la vue d'afficher le message (Game over) et lorsque on recommence une autre alors qu'elle efface ce message .

Tomber directement une forme sur le terrain :

On a ajouté une fonctionnalité dans le jeu, cela lorsque on clique sur la barre d'espace la forme tombe directement sur le terrain, donc dans le contrôleur, on a liez cet événement (clique sur <space>) avec une méthode **forme_tombe_directement(self)** qui met le **self.__delai** a 0 c.-à-d. la forme tombe directement sur le terrain .

On a ajouté les deux formes suivant au jeu :





Terrain : On a modifie le terrain de jeu et celui de la forme suivante en suppriment les entres lignes grise, donc on a rendu le terrain tout noir.