LMI 1 - Semestre 2 Mars-Avril-Mai 2017

Université d'Artois
Faculté des Sciences Jean Perrin

ALGO2 – Algorithmique et Programmation 2

Projet : Tetris

Étape 1 - Une forme tombe

La première étape sera consacrée à réaliser une version du jeu où une forme tombe toute seule sur le terrain. Lorsqu'elle se pose, une nouvelle forme se met à tomber toute seule. Le jeu s'arrête lorsque le terrain est rempli. Le joueur ne peut pas intervenir (sauf pour quitter l'application !).

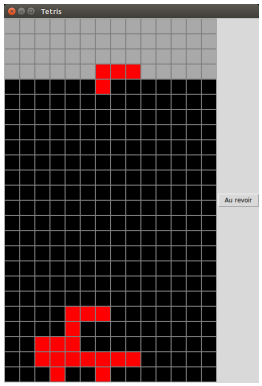


FIGURE 1 – Projet Tetris : étape 1

Le modèle

Globalement, le modèle est constitué de deux éléments : un terrain, qui est une matrice d'entiers, qui contient les carrés des formes qui se sont posées au bas du tableau, et une forme qui tombe.

Dans un premier temps, vous allez construire la classe `ModeleTetris` dans le fichier `modele.py`.

1. Le constructeur de la classe `ModeleTetris` prend en paramètre un nombre de lignes et de colonnes (vous pouvez fixer une valeur pour ces deux paramètres par défaut). Quatre lignes sont ajoutées pour le lancer de formes (ce sont les lignes grises en haut du terrain), et les attributs `self.__haut` et `self.__larg` contiennent respectivement le nombre de lignes et le nombre de colonnes du Tetris (en incluant la zone grise). L'attribut `self.__base` est fixé à `4` :c'est l'indice de la première ligne du terrain noir. L'attribut `self.__terrain` est la matrice qui représente le terrain de jeu : c'est une liste de listes d'entiers, de dimensions `self.__haut` lignes et `self.__larg` colonnes. Les valeurs initiales sont `-2` dans les quatre premières lignes et `-1` sur le reste du terrain. Une valeur négative indique que la case du terrain est vide. Un dernier attribut `self.__forme` est ainsi initialisé dans le constructeur de `ModeleTetris` :

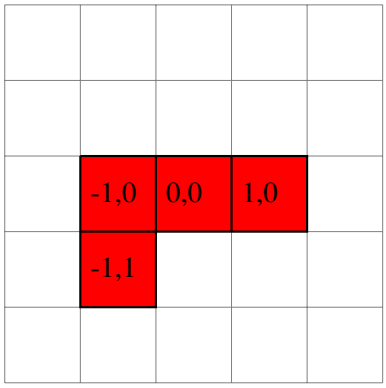
   ```
   self.__forme = Forme(self)
   ```

   Nous allons implémenter la classe `Forme` dans le paragraphe suivant. Vous voyez ici qu'une `Forme`, pour être construite, a besoin d'une instance de `ModeleTetris` en paramètre de son constructeur.

2. Implémentez les méthodes `get_largeur`, `get_hauteur` qui retournent les valeurs des attributs correspondants ;
3. Implémentez les méthodes `get_valeur` et `est_occupe`, qui, pour un numéro de ligne et un numéro de colonne donnés, retournent respectivement la valeur du terrain dans la case correspondante, ou un booléen qui indique si la case est occupée ;
4. Implémentez la méthode `fini` qui indique si la partie est finie : c'est le cas lorsqu'une case de la ligne noire la plus haute (celle d'indice `self.__base`) a aumoins une case occupée.

Nous allons maintenant construire la classe `Forme` avant de revenir à la classe `ModeleTetris`. Nous allons proposer une implémentation qui nous permettra ultérieurement de prendre assez facilement en compte des formes différentes.

1. Le constructeur de la classe `Forme` prend une instance de `ModeleTetris` en paramètre et le mémorise dans un attribut `self.__modele`. L'attribut `self.__couleur` sera initialisé dans un premier temps à `0`. Pour modéliser une forme, on considère une liste de coordonnées relatives, par rapport à une des cellules de la forme qu'on choisit de manière arbitraire (c'est cette case qui servira de pivot quand on fera tourner les formes). Prenons comme exemple la forme qui tombe dans la figure 1 :



   On va donc initialiser l'attribut `self.__forme` aux valeurs suivantes :

```
self.__forme = [(-1,1),(-1,0),(0,0),(1,0)]
```

   Enfin, on va initialiser les attributs `self.__x0` et `self.__y0` qui représenteront les coordonnées dans le terrain de la cellule `(0,0)` de la forme. `self.__y0` sera initialisé pour le moment à `0` (indice de la plus haute ligne grise du terrain), et `self.__x0` à une valeur choisie aléatoirement dans l'intervalle `[2;self.__modele.get_largeur()-2]`.
2. Implémentez la méthode `get_couleur` qui retourne la couleur de la forme ;
3. Implémentez la méthode `get_coords` qui retourne une liste de couples `(int,int)` représentant les coordonnées absolues de la forme sur le terrain du modèle ;
4. Implémentez la méthode `collision` qui retourne vrai si la forme doit se poser, faux sinon. Il y a collision lorsque pour l'une des coordonnées absolues de la forme, soit on est arrivé sur la dernière ligne la plus basse du terrain, soit on est sur une cellule juste au-dessus d'une cellule occupée sur le terrain.
5. Implémentez la méthode `tombe` qui fait tomber d'une ligne (qui change la valeur de l'attribut `self.__y0`) la forme s'il n'y a pas collision. Cette méthode retourne Vrai s'il y a eu collision et que la forme n'a pas bougé, faux sinon.

Retour à la classe `ModeleTetris` : on peut maintenant implémenter ses dernières méthodes :

1. Implémentez la méthode `ajoute_forme(self)` qui *pose* `self.__forme` sur le terrain : à chaque coordonnée absolue de `self.__forme`, on affecte la valeur de sa couleur dans le terrain ;
2. Implémentez la méthode `forme_tombe(self)` qui fait tomber `self.__forme`. Si elle n'est pas tombée (il y a eu collision), alors `self.__forme` doit être ajoutée sur le terrain, et `self.__forme` est réinitialisé à une nouvelle forme. Cette méthode retourne vrai s'il y a eu collision, faux sinon ;
3. Implémentez la méthode `get_couleur_forme` qui retourne la couleur de `self.__forme` ;
4. Implémentez la méthode `get_coords_forme` qui retourne les coordonnées absolues de `self.__forme`.

La vue

Nous allons maintenant implémenter la classe `VueTetris` dans le fichier `vue.py`. Il faut ici importer les modules `tkinter` et `modele`. Nous définirons également deux constantes : `DIM`, pour la taille d'une case du Tetris (`30` peut être une bonne valeur), et `COULEURS` qui est la liste des couleurs qui seront utilisées pour le jeu :

```
COULEURS = ["red","blue","green","yellow","orange","purple","pink",
            "dark grey","black"]
```

Pour la classe `VueTetris` :

1. Le constructeur de la classe `VueTetris` prend une instance de `ModeleTetris` en paramètre et le mémorise dans un attribut `self.__modele`. Il construit la fenêtre principale de l'application et tous les composants de la vue : pour le moment, il s'agit d'un `Canvas` à gauche, et d'un `Button` pour quitter l'application. Le `Canvas` est mémorisé dans un attribut `self.__can_terrain`, et il est dimensionné en fonction du nombre de lignes et de colonnes du modèle et de `DIM`. Le bouton pour quitter sera placé dans une `Frame` : pour le moment, il est tout seul, mais d'autres éléments seront posés ultérieurement.
2. Dans le constructeur, on va dessiner une première fois sur le `self.__canTerrain` : nous allons dessiner des rectangles colorés par la méthode `create_rectangle` de la classe `Canvas`. Cette méthode prend en arguments les coordonnées du pixel haut gauche et du pixel bas droit du rectangle, la couleur du cadre qui l'entoure (dans mes figures, j'ai choisi la couleur `"grey"`) dans le paramètre `outline`, et la couleur de remplissage dans le paramètre `fill`. La méthode `create_rectangle` retourne un objet sur lequel nous allons agir pendant le jeu : nous ne dessinerons plus de nouveaux rectangles au fur et à mesure, mais nous allons modifier la couleur de remplissage des rectangles déjà dessinés. Vous devez donc mémoriser tous ces objets dans une liste de listes conservée dans l'attribut `self.__les_cases`. Enfin, la couleur de remplissage sera choisie en fonction de la valeur contenue sur le modèle dans la case du terrain correspondante : l'entier retourné par le modèle sera utilisé comme indice de la couleur choisie dans la liste `COULEURS` ;
3. Implémentez la méthode `fenetre` qui retourne l'instance de `Tk` de l'application ;
4. Implémentez la méthode `dessine_case(self,i,j,coul` qui remplit la case en ligne `i` et en colonne `j` de la couleur à l'indice `coul`. Pour cela, vous utiliserez la méthode `itemconfigure` de la classe `Canvas`, que vous utiliserez sur `self.__can_terrain`. Cette méthode prend en premier paramètre l'objet du `Canvas` sur lequel on veut agir (ici, un élément de `self.__les_cases`), puis on peut ajouter le paramètre `fill` auquel on affecte une nouvelle couleur ;
5. Implémentez la méthode `dessine_terrain` qui met à jour la couleur de tout le terrain en fonction des valeurs du modèle ;
6. Implémentez la méthode `dessine_forme(self,coords,couleur)`, où `coords` est une liste de couples `(int,int)` et `couleur` un entier, et qui remplit de `couleur` les cases dont les coordonnées sont données dans `coords`. Cette méthode permet de faire apparaître une forme sur le terrain.

Le contrôleur

Il ne nous reste plus qu'à écrire le contrôleur, qui se trouvera dans un module séparé cette fois-ci, et le script principal qui lance l'application.

La classe `Controleur` est créée dans un fichier `pytetris.py`. Vous pouvez importer tout de suite les deux autres modules que vous avez créés, `modele` et `vue`. Vous allez en plus importer un nouveau module, le module `time`.

Nous commencons par la fin. Le script principal sera également présent dans ce fichier. Il vous est donné :

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

Passons maintenant à la classe `Controleur` :

1. Comme on le voit dans le script principal, le constructeur 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 modèle. Il le conserve également dans un attribut. Il récupère auprès de la vue l'instance de la fenêtre `Tk` qui contient l'application et la mémorise dans un attribut `self.__fen`. Il demande à la vue de dessiner la forme du modèle (attention, vous avez déjà défini tous les éléments nécessaires pour faire cette action). Vous devez encore faire trois actions dans le constructeur :
   — créez un attribut `self.__delai` que vous initialiserez à `320` (vous pourrez fixer la valeur comme vous le souhaitez). Cela sera le nombre de millisecondes d'attente à chaque étape de la descente d'une forme ;
   — appelez la méthode `self.joue()` de la classe `Controleur` dont le code vous est donné ensuite ;
   — lancez la boucle d'écoute des événements sur `self.__fen`
2. Implémentez la méthode `joue` qui vous est donnée :

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

   Cette méthode teste si la partie est finie, puis appelle la méthode `affichage` de la classe `Controleur`. Ensuite, la méthode `after` de `Tk` permet de rappeler à nouveau cette méthode `joue` au bout de `self.__delai` millisecondes d'attente ;
3. Implémentez la méthode `affichage` de la classe `Controleur` : le contrôleur indique au module qu'il doit faire tomber la forme, puis il demande à la vue de redessiner son terrain, puis de redessiner la forme.

Testez votre application. Bravo, c'est fini pour cette étape !