LMI 1 - Semestre 2 Mars-Avril-Mai 2018

Université d'Artois
Faculté des Sciences Jean Perrin

ALGO2 – Algorithmique et Programmation 2

Projet : Same

Préambule : Préparez vos outils de travail. Prévoyez un répertoire pour votre projet, et dans ce répertoire créez quatre répertoires `etape1`, `etape2`, `etape3` et `etape4`. Pour plus de simplicité, créez dans chacun de ces répertoires un répertoire `img` et copiez-y les images des billes mises à votre disposition sur Moodle.

Étape 1 - On affiche le terrain et on supprime une bille

La première étape sera consacrée à réaliser une version du jeu qui nous permettra d'installer tous les éléments de base. Le terrain de jeu est affiché, et en cliquant sur une bille on la supprime (et les autres restent en place). On peut ré-initialiser une nouvelle partie.

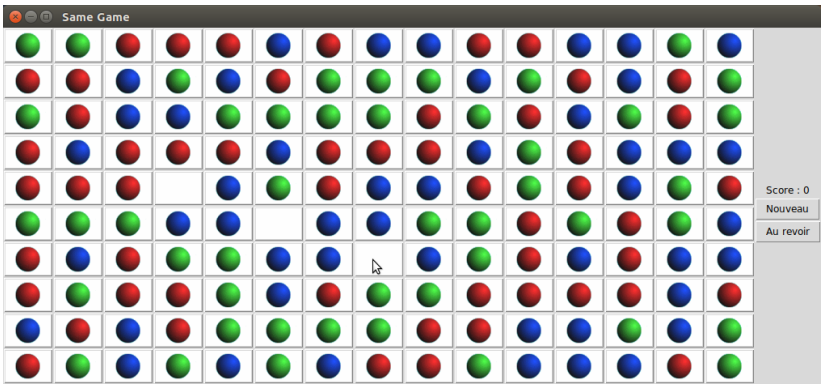

FIGURE 1 – Projet Same : étape 1 - on a supprimé 3 billes.

Le modèle

Globalement, le modèle est constitué de deux éléments : une case, qui, pour cette étape, ne contiendra que la couleur de la bille qu'elle contient, qui sera représentée par un entier, et le terrain, qui est une matrice de cases.

Les classes `Case` et `ModeleSame` décrites ci-dessous doivent être définies dans le fichier `modele.py`.

La classe `Case`

Vous allez créer la classe `Case` qui modélise une bille du jeu Same.

— Le constructeur de la classe `Case` prend en paramètre la couleur de la bille, et qui est représenté par un entier positif ou nul (la valeur `-1` sera réservée aux cases vides). La couleur est conservée dans un attribut `self.__couleur`.

— La classe `Case` possède un certain nombre de méthodes très simples que vous devez spécifier et implémenter

  — une méthode `couleur` qui retourne la couleur de la bille ;

— une méthode `change_couleur` qui prend en paramètre une valeur entière et change la couleur de la bille ;
— une méthode `supprime` qui enlève la bille de la case (passe la couleur à `-1`) ;
— une méthode `est_vide` qui indique si la case est vide.

Pour le moment, cette classe est très simple, mais elle évoluera par la suite.

La classe `ModeleSame`

Nous nous occupons maintenant du modèle pour le jeu Same qui sera défini dans la classe `ModeleSame`.

1. Le constructeur de la classe `ModeleSame` prend en paramètre un nombre de lignes, un nombre de colonnes et un nombre de couleurs. Choisissez des valeurs qui seront positionnées par défaut : en géénrale, le jeu se joue soit en 15 lignes et 20 colonnes, avec 4 couleurs ; soit en 10 lignes, 15 colonnes et 3 couleurs. Chacun de ces paramètres sera conservé dans un attribut (respectivement `self.__lig`, `self.__nbcol` et `self.__nbcouleurs`). L'attribut principal de `ModeleSame` est `self.__mat`, une matrice de `Case` (une liste de `__nblig` listes de `__nbcol` instances de `Case`). Créez `self.__mat`. Les couleurs des cases seront choisies aléatoirement dans l'intervalle `[0;__nbcouleurs - 1]`. Un dernier attribut est `self.__score` qui contiendra le score du joueur et sera initialisé à `0` ;
2. Le modèle possède quelques méthodes très simples pour cette étape (que vous devez spécifier et implémenter) :
   — les méthodes `score`, `nblig`, `nbcol`, `nbcouleurs` qui retournenent, chacune, la valeur de l'attribut de même nom ;
   — une méthode `coords_valides` qui prend deux paramètres `i` et `j` et qui indique s'il s'agit de coordonnées valides pour le jeu ;
   — une méthode `couleur` qui prend deux paramètres `i` et `j` et qui retourne la couleur de la bille en `(i,j)` ;
   — une méthode `supprime_bille` qui prend deux paramètres `i` et `j` et qui supprime la bille en `(i,j)` ;
   — une méthode `nouvelle_partie` qui ré-initialise toutes les cases en changeant leur couleur (par une couleur choisie aléatoirement dans l'intervalle des couleurs possibles). Il n'y a plus de cases vides après l'appel à cette méthode.

La vue

Nous allons maintenant implémenter la classe `VueSame` dans le fichier `vue.py`. Il faut ici importer les modules `tkinter` et `modele`.

Afficher le jeu, réinitiaiser une partie

Pour la classe `VueSame` :

1. Le constructeur de la classe `VueSame` prend une instance de `ModeleSame` en paramètre et le mémorise dans un attribut `self.__same`. Il construit la fenêtre principale de l'application et tous les composants de la vue : il s'agit d'une grille de `Button` à gauche, et d'un `Label` (pour l'affichage du score), d'un `Button` pour démarrer une nouvelle partie et d'un autre pour quitter l'application.

   Après avoir créé la fenêtre principale, initialisez dans un attribut `self.__images` toutes les images des billes que vous voulez utiliser. Vous avez le choix entre trois tailles d'images : `grand`, `medium` et `petit`. Choisissez celle que vous préférez pour le rendu visuel. On n'utilisera pas les `sphereXblack` pour le moment. Par contre, positionnez la `spherevide` en dernier dans votre liste d'images.

Les `Buttons` affichés dans la grille (pour représenter les billes), seront mémorisés dans une matrice de `Button`, qui sera l'attribut `self.__les_btns`. Chaque bouton en position `(i,j)` affiche l'image qui se trouve dans `self.__images` à l'indice donné par la couleur de la bille en `(i,j)` dans le modèle.

La dernière action du constructeur et de lancer la boucle d'écoute des événements (`mainloop()`).

2. Dans un premier temps, spécifiez puis écrivez les deux méthodes suivantes :
   — la méthode `redessine` parcourt tous les boutons de `self.__les_btns` et change l'image qu'ils affichent en fonction de `__same` ;
   — la méthode `nouvelle_partie` qui est associée au bouton *Nouveau* : elle doit demander au modèle de réinitialiser une nouvelle partie, puis doit mettre à jour l'affichage.

Vous pouvez maintenant écrire le script principal de votre application, qui ne devrait plus bouger par la suite :

```
if __name__ == "__main__" :
    # création du modèle
    same = modele.ModeleSame()
    # création de la vue qui créé les contrôleurs
    # et lance la boucle d'écoute des évts
    vue = VueSame(same)
```

Testez votre application : elle doit afficher un jeu initial, puis permettre (bouton *Nouveau*) de rénitialiser la partie, et enfin (bouton `Au revoir`) de quitter l'application.

Supprimer une bille

C'est la dernière partie de cette étape. Vous allez associer à chaque click sur un bouton de la grille l'action de supprimer une bille.

Spécifiez puis écrivez la méthode `creer_controleur_btn(self,i,j)` qui retourne une fonction. La fonction retournée (que vous devrez spécifier) s'appelle `controleur_btn()` et demande au modèle de supprimer la bille en `(i,j)`, puis demande à la vue de se redessiner.

Maintenant, juste après avoir créé un bouton de la grille en `(i,j)`, vous associerez à l'attribut `command` de ce bouton le contrôleur associé.

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

Il vous ne reste plus qu'à préparer l'étape suivante : copiez vos fichiers `modele.py` et `vue.py` dans le répertoire `etape2`. Déposez dans un fichier au format `.zip` (ou `.tgz`, mais attention, les `.rar` sont exclus et ne seront pas corrigés) votre étape 1. Maintenant, vous êtes prêt.e.s pour la suite !