Implémentation en scala d'un tetris, avec intelligence artificielle

Simon Andreux, Julien Gamba, Clément Schreiner

2013-12-08

Contents

1 Introduction - Pourquoi Scala ?

Nous n'aimons pas Java, notamment pour la verbosité du code.

Clément, ayant un peu d'expérience en programmation fonctionnelle, a proposé scala, qui a l'avantage d'être entièrement objet, mais aussi de proposer des fonctionnalités qu'on trouve généralement dans les langages fonctionnels tels que le pattern matching et l'inférence de type.

Il est de plus entièrement intéropérable avec Java, peut importer directement ses bibliothèques, et se compile en bytecode exécutable par la JVM.

2 Modélisation, structures de données

3 Interface réseau

Pour assurer la communication entre l'IA et la partie graphique du projet, nous avons choisi de passer par un serveur TCP.

Scala propose plusieurs bibliothèques. Nous avons retenu `Netty` pour ce projet. Elle est assez standard, et est donc très bien documentée. Elle utlise des bibliothèques Java, ce qui impose d'utiliser des méthodes Java directement dans le code Scala.

En soi, ce n'est pas très contraignant, mais cela peut poser certains problèmes dans certains cas. Par exemple, dans notre programme, un fois le serveur lancé, il est possible pour l'utilisateur – et plus particulièrement pour l'IA – d'envoyer des messages au serveur directement sous forme de `String`. Ces `String`s passent par un buffer (un `BigEndianHeapChannelBuffer`), qui encode les `String`s et les rend illisibles sans traitement préalable. Cela impose l'utilisation d'un handler qui va agir au niveau de la `PipelineFactory` pour décoder les flux et les rendre utilisables.

4 Problèmes rencontrés

4.1 Récupérer une ligne dans la matrice de la grille

L'ensemble des blocs de la grille (excepté la pièce courante, qui n'est pas fixe) est représenté par une matrice, implémentée sous forme d'Array d'Array.

Ainsi, pour une matrice de 3x3:

```
scala> import Array.ofDim
scala> var blocks = ofDim[Boolean](3, 3)
blocks: Array[Array[Boolean]] = Array(Array(false, false, false),
  Array(false, false, false),
  Array(false, false, false))
scala> val b = blocks(0)(2)
b: Boolean = true
```

b correspond ici au bloc d'abscisse 0 et d'ordonnée 2, et *blocks(0)* à la première colonne

Pour vérifier la présence de lignes complètes, on a besoin de récupérer une ligne. Nous avons trouvé une solution fonctionnelle et élégante utilisant *map*:

```
def getLine(blocks) = blocks.map(_(y))
```

4.2 Définir simplement les formes des pièces

Après plusieurs tentatives plus moins élégantes, nous avons finalement choisi une solution assez simple. Chaque forme est définie par un objet implémentant le trait ShapeKind, qui comport deux données: la couleur, et une liste de coordonnées relatives des blocs la composant, pour chaque orientation possible. Concrètement: une liste de liste de paires d'entiers.

5 Intelligence artificielle

5.1 Fonctionnement

L'Intelligence Artificielle a été conçu en bottom-up. La classe AI s'occupe de générer une liste exhaustive de toutes les grilles possibles pouvant être obtenues à partir d'une seule forme. Pour cela, elle crée des copies à gauche, à droite (fonction reach) de la grille pour la forme initiale.

Ensuite, elle effectue cette opération pour toutes les rotations possibles (fonction listGrid et listGridWithRot) de la forme actuelle de la grille. Enfin, l'ia calcule un score pour chaque grille (méthode eval) et renvoit la grille qui a réalisé le meilleur score en utilisant un algorithme map/reduce.

5.2 Evaluation

L'évaluation a consisté à attribuer un score à chaque grille en fonction de différents critères:

1. la hauteur des blocs
2. le nombre de trous dans la grille
3. les blocs considérés comme genant (au-dessus d'un trou)
4. la densité des blocs (nombre de voisin, etc)

5.3 Améliorations possibles

Il est possible d'évaluer les blocs différemment en fonction de leur type.

Par ailleurs, les différentes valeurs utilisées pour paramétrer l'IA ne sont pas excellentes, il serait possible de concevoir un algorithme génétique distribué afin de les calculer.

6 Organisation

Nous avons réparti le travail ainsi :

- Simon : intelligence artificielle
- Julien : début de l'interface réseau entre la GUI et l'AI
- Clément : interface graphique, interface réseau
- Simon et Clément : structures de données représentant le jeu

Nous avons utilisé git pour faciliter la collaboration.