Projet Phineloops : Documentation Développeur

Betty Bismuth et Jean Clemenceau

Université Paris-Dauphine

M1 MIAGe

Java Avancé

Janvier 2018

Sommaire

1 Description de l'architecture

Le projet respecte le modèle de conception MVC (Modèle Vue Contrôleur). Nous justifions ce choix par la volonté de développer une interface graphique dés le commencement du projet.

Le package model comprend les classes Piece et Grid permettant respectivement de modéliser chaque pièce et une grille contenant des pièces.

Nous avons également ajouté une classe énumération PieceProperties qui permet :

- De retrouver grâce à l'identificateur d'une pièce des caractéristiques arbitraires dépendantes des informations imposées d'éviter la redondance dans le code
- De séparer les traitements logiques sur les pièces de la récupération pure et simple de données informatives telles que les correspondances entre un couple (numéro_pièce, orientation_pièce) et la forme visuelle de la pièce, ou encore le nombre de connexions sur un côté d'une pièce suivant son orientation.
- D'ajouter facilement une nouvelle pièce en apportant ses données dans cette classe énumération

Le package view comporte toutes les classes pour l'affichage des éléments de l'interface graphique (disposition et apparence des panels, des boutons...) ainsi que la logique pour afficher les données provenant du modèle (grille et pièces qui lui sont associées) et les possibilités d'actions sur la grille affichées (appels aux fonctionnalités du packages programs). Nous avons mis en place le patron de conception Observateur pour que l'affichage d'une pièce qui est donc notre observateur (PieceDisplay.java) soit mis à jour automatiquement lorsque son sujet observé (Piece.java) pivote.

Le package controller se charge de traiter les actions de l'utilisateur qui impactent directement les classes du package model, dans notre cas l'utilisateur interagit directement uniquement avec Piece.java via des clics sur les éléments affichant les pièces, nous avons donc une seule et unique classe de contrôle : ClickOnPieceController.java

Le package programs contient les classes permettant d'effectuer différents traitements automatiques pour interagir avec une grille du jeu. Elles sont toutes sans attributs et avec uniquement des méthodes statiques. Ce choix a été effectué car ces classes ne représentent pas des objets que l'on souhaiterait instancier mais des actions sur une grille.

La classe Main permet le déroulement du jeu. Elle récupère les arguments passés en ligne de commande ou par l'intermédiaire de l'interface graphique et

crée en réponse soit un Generator, soit un Checker, soit un Solver, puis appelle leurs méthodes.

Figure 1: Diagramme de classes

2 Organisation du travail et difficultés d'implémentation

Nous avons travaillé en binôme sur l'intégralité du projet.

Nous avons commencé par nous réunir pour réfléchir à l'architecture et à la manière de modéliser le jeu. Puis nous avons commencé à coder la plus grande partie du package model, ensuite nous avons travailler sur les programmes en commençant par le générateur et le vérificateur sans grandes difficultés, nous avancions en parallèle sur l'interface graphique qui ne comportait pas de réflexions algorithmiques poussées.

C'est en travaillant sur le solveur que nous avons rencontré le plus de difficultés.

Nous voulions tout d'abord réfléchir à l'algorithme et à son déroulement, et c'est surtout cette compréhension qui fut compliquée. En effet, nous arrivions rapidement à un algorithme qui pouvait tester si une pièce était correctement positionnée ou s'il fallait la pivoter mais dès qu'il fallait revenir en arrière dans l'arbre de recherche, nous rencontrions des erreurs. Nous avons fait plusieurs versions de l'algorithme avant de comprendre comment simuler la récursivité à l'aide d'une pile de positions.

3 Fonctionnalités

Le vérificateur parcourt la grille de gauche à droite et vérifie pour chaque pièce si son orientation est compatible avec la grille (pas de connexions dans le vide) mais aussi avec ses voisins du haut et de la gauche (le bon nombre de connexions avec eux).

Le générateur choisit la première pièce au hasard puis pour les pièces suivantes de gauche à droite, recherche les pièces aux orientations possibles pour cet emplacement et choisit de nouveau au hasard parmi les possibilités.

Le solveur permet une résolution quasi-exhaustive avec deux choix de pièces. La première méthode solveRandom choisit au hasard la pièce par laquelle commencer la résolution puis avance dans la grille de gauche à droite. La seconde méthode solveFix récupère toutes les pièces fixées de la grille. Ces pièces sont les pièces vides et les croix c'est-à-dire celles qu'il n'est pas nécessaire de tester. Elle va ensuite prendre une de ces pièces, récupérer ses voisins et trouver leurs orientations optimales. Puis la méthode effectue le même traitement avec tous

les voisins de toutes les pièces fixées et termine les pièces restantes de gauche à droite. Nous n'avons pas pu coder le solveur en multi-thread, et ce par manque

de temps. En effet, nous avons travaillé des dizaines d'heures sur l'algorithme de base du solveur et pendant un long moment sans succès ce qui a retardé notre progression.

L'interface graphique permet d'afficher une grille proprement. L'utilisateur peut générer une nouvelle grille de la taille souhaitée, il peut cliquer sur les cases afin de faire pivoter les pièces, vérifier si sa solution est bonne et demander à ce que la grille soit résolue. Il est d'ailleurs possible de voir le solveur travailler.

Toutes les structures ont été crées de manière générique. Ainsi, ajouter des pièces présentant plus de 4 connexions, comme dans le version HEX du jeu, ne devrait pas créer d'erreur, les algorithmes de résolution prenant en compte le nombre de connexions possibles.

Si l'interface graphique n'est pas utilisée, un affichage de débogage en caractères Unicode apparaît en ligne de commande.