Documentation du développeur

Présentation des classes

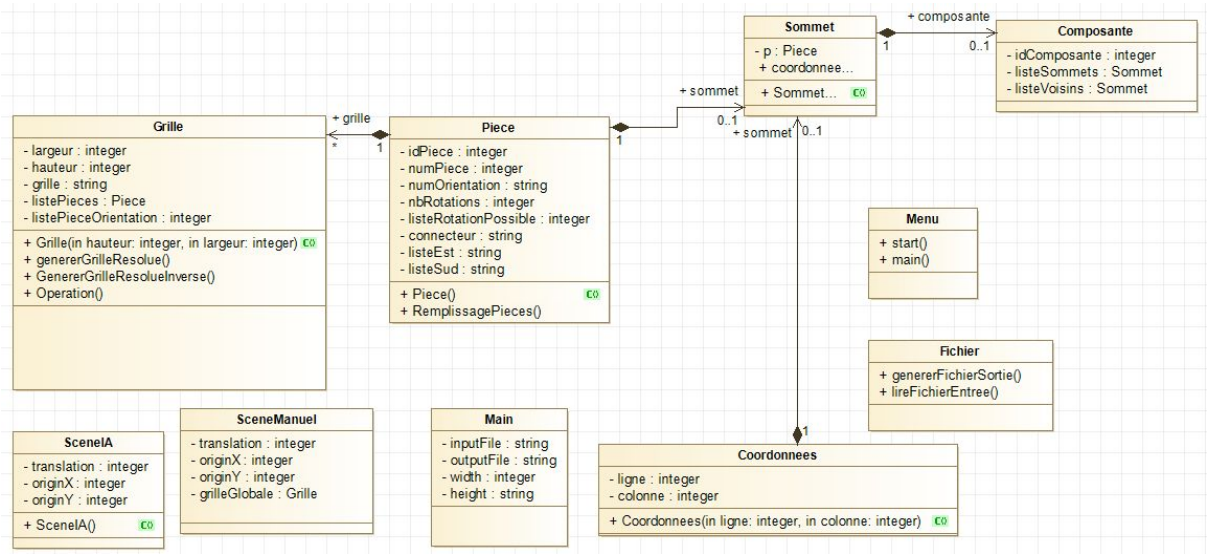
Les différentes classes utilisées sont Pièce, Grille. Ces classes représentent de manière abstraite les éléments qui permettent de caractériser le jeu. La pièce est un objet qui est caractérisé par les attributs suivants (tous en prives)

int idPiece : permet d'identifier de façon unique la pièce.
int numPiece : permet de définir la pièce indépendamment de ses rotations.
int numOrientation : permet de déterminer la rotation d'une pièce bien définie.
int nbRotations :représente le nombre de rotations possibles pour une pièce donnée.
int [] rotationPossible :pour chaque pièce, ses différentes rotations possibles
String connecteur : les connecteurs existants pour une pièce donnée.
int listeEst[] : liste des pièces susceptibles d'être rattachées à la droite de la pièce courante.
int listeSud [] :liste des pièces susceptibles d'être rattachées en bas de la pièce courante.

La grille est un objet caractérisé par les attributs suivants

int n : représente la longueur de la grille.
int m : représente la largeur de la grille.
Piece grille [][] : représente une matrice de Pièces.

Diagramme de classes



Présentation du modèle MVC relatif à l'implémentation

Dans notre implémentation, nous avons eu recours à la représentation selon le modèle MVC, en effet, nous avons compartimenté nos classes java selon leurs rôle.

Modèle : représente les classes qui définissent l'élément abstrait qu'on souhaite manipuler, ici il s'agit des classes Pièce, Composante, Sommet, Coordonnées et Grille.
Vue : représente la partie visible de l'application, elle est représentée par les classes SceneIA, SceneManuel et Menu.
Contrôleur : Module qui traite les actions de l'utilisateur, modifie les données du modèle et de la vue.
Dans notre conception, la partie contrôleur est représentée par les méthodes

```
imageView.setOnMouseClicked(new EventHandler<MouseEvent>(){});
```

Ces méthodes permettent de faire des traitements après que l'utilisateur ait cliqué sur imageView.

Génération d'une grille
Afin de générer une grille qui admet au moins une solution, nous avons procédé de la manière suivante :
Nous avons dans un premier temps généré une grille résolue contenant un nombre quelconque de composantes connexes.
Pour ce faire, nous avons considéré au début l'ensemble des pièces pouvant se trouver à l'extrême haut-ouest ensuite, nous avons généré une pièce aléatoirement à sa droit et qui soit compatible avec cette dernière, deux situations se présentent : soit la pièce de gauche présente un connecteur vers la droite, alors il faudra que la pièce se trouvant à la droite ait un connecteur vers gauche, la deuxième situation consiste à ne pas avoir de connexion entre ces deux pièces.
On réitère ensuite ce processus pour toutes les pièces situées à la première ligne de la grille.

Remarque
Le choix d'une pièce se fait sur un ensemble de pièces qui sont préalablement admissibles.
Cette méthode est plus judicieuse et optimale qu'une solution naïve consistant à générer une pièce aléatoire parmi toutes les pièces existantes et ensuite tester sa compatibilité.
En effet, dans notre implémentation, quand on génère une solution aléatoire, on est sûr que cette dernière est compatible avec ses voisins déjà générés.

Concernant les autres lignes, il suffira de considérer le fait que la pièce se trouvant en dessus de la pièce actuelle, présente un connecteur vers le bas, donc la pièce courante devra présenter un connecteur vers le haut pour assurer la connexion.

L'implémentation de cette méthode se base sur l'intersection de l'ensemble des pièces compatibles avec la pièce se trouvant en haut de la pièce courante avec l'ensemble des pièces compatibles avec la pièce se trouvant à gauche de la pièce courante.

Cette intersection représente l'ensemble des pièces compatibles avec les deux contraintes citées précédemment.

A la fin de ces traitements, on obtiendra une grille totalement résolue, qui sera une solution potentielle (mais pas la seule car pour une grille donnée, on peut avoir plus d'une solution).
Il suffira d'effectuer des rotations sur les pièces générées de manière aléatoire pour obtenir à la fin la grille souhaitée.

Tester si une grille est résolue
Dans le but de tester si une grille est résolue, il suffit de tester si toutes les pièces sont bien positionnées, en effet, pour savoir si toutes les pièces sont bien positionnées, on teste si la pièce courante est compatible avec la pièce se trouvant à sa droite et son haut uniquement.
Pour les pièces se trouvant aux extrémités, on considère que la pièce qui lui est adjacente est une pièce vide, ce qui nous permet de faire un algorithme compacte et générique qui fait que 2 tests au lieu de 4.
En effet, il n'aurait pas été très judicieux de tester la position de chaque pièce
par rapport à ses 4 pièces voisines car il y aurait des redondances.

Résolution d'une grille
Utilisation d'une heuristique lors de la résolution
Concernant la résolution d'une grille, nous avons tout d'abord pris le soin d'implémenter une heuristique permettant de tester si la grille admet une solution.
Une grille n'admet pas de solution si par exemple la pièce 12 (+) se se trouve à la 1ère ligne ou colonne ou bien à la dernière ligne ou colonne.
Ainsi que de faire des tests sur la parité du nombre de connecteurs total.
Si le nombre total de connecteurs est impair, on retourne directement un false pour indiquer que la grille correspondante n'admettra jamais de solutions.
Ces heuristiques nous permettent d'éviter de faire des parcours inutiles et cela en testant la grille sur des conditions nécessaires mais pas suffisantes.
Par exemple : le fait qu'une grille contienne un nombre pair de connecteurs n'implique pas directement que cette dernière admet au moins une solution.
Alors que si le nombre est impair, alors on est sûr que la grille n'admet pas de solutions.

Principe de résolution d'une grille
Pour la résolution, nous avons appliqué un algorithme de recherche en profondeur qui
fonctionne de la manière suivante :
On empile dans un premier temps l'objet grille courant et on sauvegarde les coordonnées de la première pièce à traiter.
Tant que la pile ne sera pas vide, on dépile la grille se trouvant au sommet de la pile, on teste si cette dernière représente une solution au problème initial. Si ce n'est pas le cas, on empile toutes les rotations possibles ainsi que les coordonnées correspondantes.
à la fin, deux situations se présentent :

Si la grille admet une solution, alors cette dernière sera trouvée et la valeur booléenne true sera retournée.
Si la grille n'admet aucune solution, la pile sera vide car toutes les combinaisons auront été parcourues dans notre arbre de recherche et la valeur false est retournée.

Calcul de la complexité de l'algorithme de résolution présenté
Soit l'arbre de recherche A considéré dans notre algorithme.*

- b est le facteur de branchement de l'arbre.
- d est la profondeur de la solution la moins profonde.
- m est la profondeur maximale de l'arbre de recherche.

La complexité en temps est égale à $O(b^m)$.
La complexité en mémoire est égale à $O(b * m)$.

Tester si une rotation est admissible
Nous avons choisi d'implémenter une fonction permettant de tester si la rotation d'une pièce donnée est admissible.
Elle prend en paramètres une pièce ainsi que son numéro de ligne et colonne.
Pour ce faire, il suffit de considérer les pièces se trouvant à gauche et en haut de la pièce courante.
Pour qu'une pièce soit admissible, il faut que si la pièce se trouvant en haut (resp. à gauche) présente un connecteur vers le bas (resp. vers la droite) alors la pièce courante doit présenter un connecteur vers le haut (resp. la gauche).
Concernant les pièces se trouvant aux extrémités, c'est à dire celles qui sont à la première ou dernière ligne (resp. colonne).
seront comparées à des pièces vides (pièce 1 dans notre modélisation) et celà pour permettre un algorithme plus compacte et avec moins de tests.

Remarque
Les tests par rapport à l'admissibilité prennent en compte uniquement 2 pièces adjacentes à la pièce courante ce qui nous permet de faire 2 tests au lieu de 4. donc un gain de 2 test à chaque itération.
Nous avons utilisé des iterators lors du parcours des listes pour offrir des temps d'exécutions plus performants.

Résolution du problème inverse
Nous avons choisi d'implémenter l'option du jeu inverse.
Ce jeu consiste à générer une grille et la résoudre de telle sorte à ce que toutes les pièces de cette grille soient déconnectées.
Pour ce faire, nous avons tout d'abord généré une grille résolue en appliquant la même méthode de génération expliquée ci-dessus.
À la fin, on obtient la liste des pièces compatibles avec la pièce se trouvant en haut et la liste des pièces compatibles avec la pièce se trouvant à gauche.
On génère le complément du premier ensemble, ainsi que le complément du deuxième ensemble, ce qui nous permet d'obtenir une pièce qui n'est jamais connectée à la pièce du haut et de sa gauche.

Concernant la résolution d'un tel problème, il s'agit de résoudre de la même problème en considérant la non-connectivité au lieu de la connectivité d'une pièce par rapport à ses voisines.

Génération d'une grille avec nombre de composantes connexes

Il est possible de générer une grille avec un certain nombre de composantes connexes.

La démarche suivie consiste à positionner dans un premier temps deux pièces reliées (une composante connexe comporte au moins deux pièces reliées) de façon aléatoire dans la grille et cela pour chacune des composantes connexes.

Pour chaque composante connexe, nous avons la liste des sommets qu'elle contient et la liste des voisins qui seront susceptibles d'être intégrés dans la composante.

Tant que la liste des voisins d'une composante connexe n'est pas vide, on ajoute ce sommet à la liste des sommets et on le supprime de la liste des voisins des autres composantes connexes.

Interface graphique

Nous avons conçu une interface graphique afin de permettre à l'utilisateur de visualiser directement les grilles générées et de pouvoir visualiser en temps réel la résolution d'une grille donnée par l'intelligence artificielle (résolution automatique) ou bien de lui permettre de résoudre une grille.

Nous avons choisi d'implanter l'interface graphique en utilisant la bibliothèque graphique JavaFx qui permet de réaliser diverses applications en 2D ou 3D et d'intégrer les médias vidéo et audio.

Concernant la résolution automatique, nous avons pris le soin d'intégrer des Timeline permettant de régler l'affichage des images des pièces, et cela afin de pouvoir visualiser de façon dynamique l'intelligence artificielle entrain de résoudre le problème.

Concernant la résolution manuelle, nous avons assigné à chaque imageView relative à une pièce, un setOnMouseClicked qui permet de faire du traitement, lorsque l'utilisateur clique sur l'image.

Répartition du travail

Victor s'est occupé du manuel utilisateur et de la génération de la grille, il s'est donc chargé de développer une partie du côté "modèle" (MVC) du projet.

Amine s'est occupé de la résolution de la grille, de l'interface graphique, du manuel développeur ainsi que le découpage du projet en modèle MVC.

Difficultés rencontrées

Notre principale difficulté reside dans notre collaboration via GitHub: en effet comme nous debutons sur GitHub, il était difficile pour nous de collaborer à distance sans avoir de conflits sur notre travail. Mais a force de pratiquer l'utilisation de Git, nous avons pu mieux synchroniser notre travail.

Avant d'entamer l'implémentation, il a été assez difficile d'établir une bonne modélisation du problème et qui permettrait d'effectuer tous les traitements demandés sans avoir à modifier à chaque fois la conception ou même des éléments relatifs au code source.

Nous avions aussi rencontré des difficultés au niveau du solveur qui nécessitait une bonne gestion de la pile pour éviter des débordement de la pile et de parcourir des sous-arbres inutilement.
Concernant la génération d'un nombre limité de composantes connexes, nous avions eu comme difficulté l'implémentation d'un algorithme de génération d'une grille pouvant prendre en paramètre le nombre de composantes connexes maximum et de générer ces dernières sans qu'elles se chevauchent ou qu'elles soient entrecoupées par d'autres composantes.
Par rapport à l'interface graphique, nous avions eu la difficulté de générer une interface affichant de façon dynamique les grilles.