Rapport projet : Jeu assemblage de pièces
Méthodes de conception

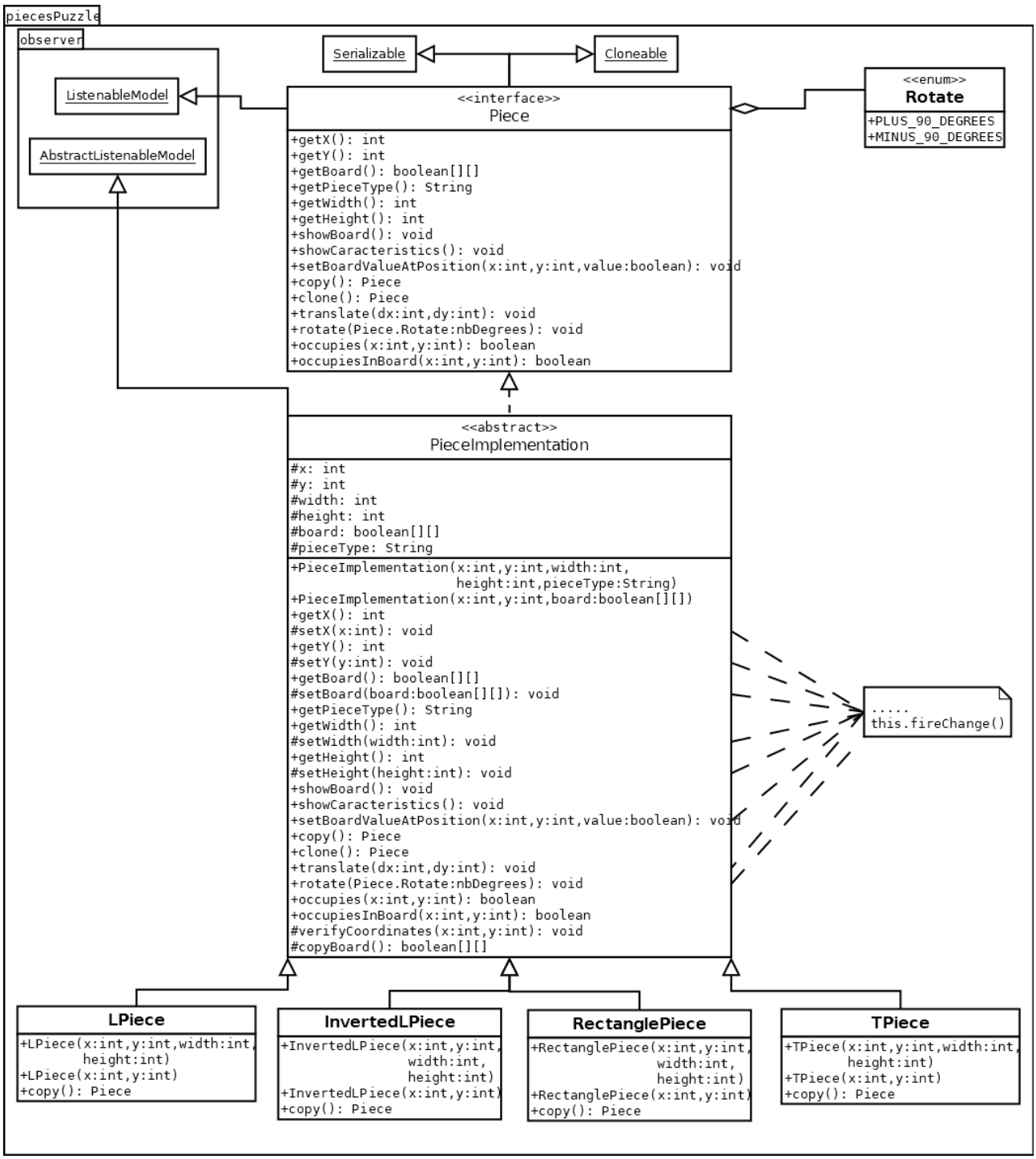
Membres du groupe :
- Arthur BOCAGE 21806332
- Guillaume LETELLIER 21804030
- Corentin PIERRE 21803752
- Alexandre PIGNARD 21701890

Informations utiles à la compréhension du logiciel :
L'application que nous avons réalisé respecte le pattern MVC (plutôt M-VC) comme demandé dans le sujet, nous avons essayé d'implémenter des patterns design dès que l'occasion se présentait (plus d'informations dans une des parties suivantes).
Au niveau des tests unitaires, il nous a été demandé de ne tester que la partie modèle de l'application. Dans le package 'piecesPuzzle', chaque pièce est testée de manière programmée et un test du pattern Observer se trouve aussi ici. Dans le package 'jeuAssemblage.model', pour les classes de ce package, nous testons les deux mais dans chacun de ses sous-packages, nous ne faisons qu'une classe de test car ses sous-package implémente des pattern Strategy, ce qui nous permet donc de tester toutes les stratégies de manière programmée.

Comment jouer au jeu :
Pour jouer au jeu d'assemblage de pièces, vous devrez tout d'abord choisir entre choisir une ancienne partie et en créer une nouvelle.
Ensuite, vous devrez sélectionner les pièces avec le clic gauche de votre souris ou mousepad afin de pouvoir visualiser la pièce sélectionnée une fois que le second clic est était réalisé. Vous pourrez bouger les pièces en déplaçant la souris et les tourner avec les touche 'a' (- 90°, tourne vers la gauche) et 'e', (90°, tourne vers la droite). Si vous ne souhaitez plus effectuer une action sur la pièce, vous pouvez la replacer à son endroit initial avant la sélection avec la touche 'Echap'. Vous pouvez vous servir du tableau de pièces disponible en bas de l'interface afin de sélectionner une pièce particulière en cliquant seulement sur la ligne souhaitée. Cette action recentre le focus sur la plateau afin de pouvoir effectuer les actions dîtes plus haut.
Puis, sur le panneau de contrôle se trouvant à droite de l'interface, vous pourrez lancer l'intelligence artificielle jouer à votre place.
Enfin, sur ce même panneau, une fois que vous souhaitez terminer la partie, vous devez entrer un nom du joueur et ainsi, valider le plateau pour calculer le score. Vous pourrez ensuite enregistrer la configuration initiale, le score, le joueur et le nombre de coups utilisés si le score est meilleur que celui avant (si vous avez choisi de charger une ancienne configuration).
N.B. : l'algorithme d'intelligence artificielle implémentée est particulièrement long lorsqu'il a beaucoup de pièces sur celui-ci et lorsqu'il ne reste plus beaucoup de jouer à jouer.

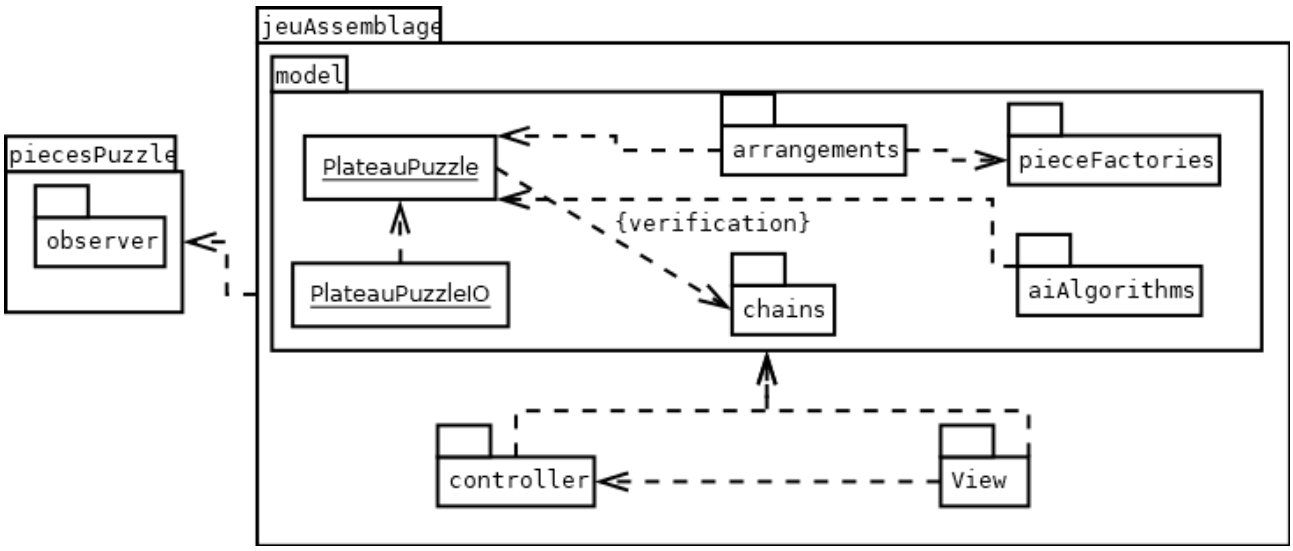
Diagrammes de classes (et de packages) et explications :
Nous ne présenterons pas le diagramme de classes pour 'piecesPuzzle.observer' car nous avons réutilisé celui donné en CM et TP.

Voici le diagramme de classes pour le package 'piecesPuzzle' :



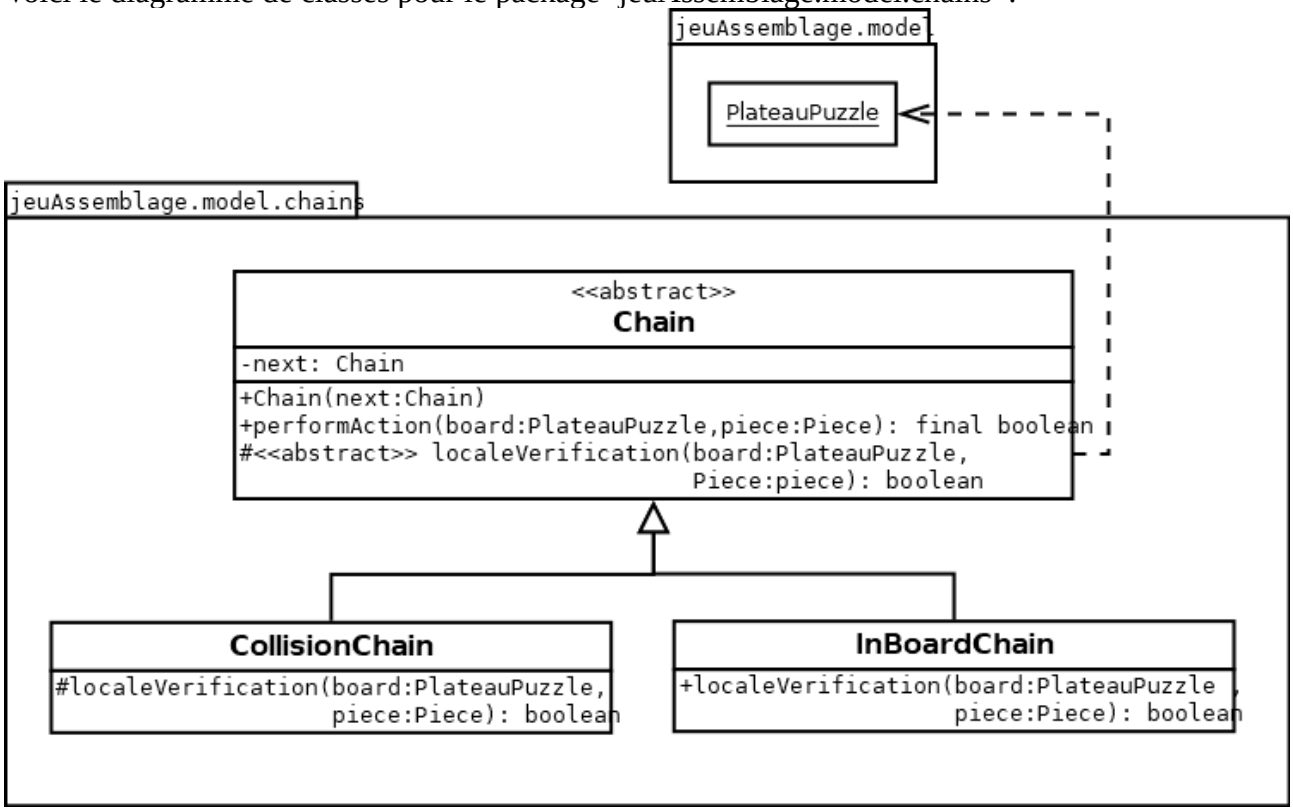
Ce package définit un certain nombre de pièces pouvant être utilisées dans différents jeux de pièces et intégrant le pattern Observer afin de que chaque pièces puissent informé si un changement est effectué sur celle-ci.

Voici le diagramme de package pour le package ‘jeuAssemblage’ :



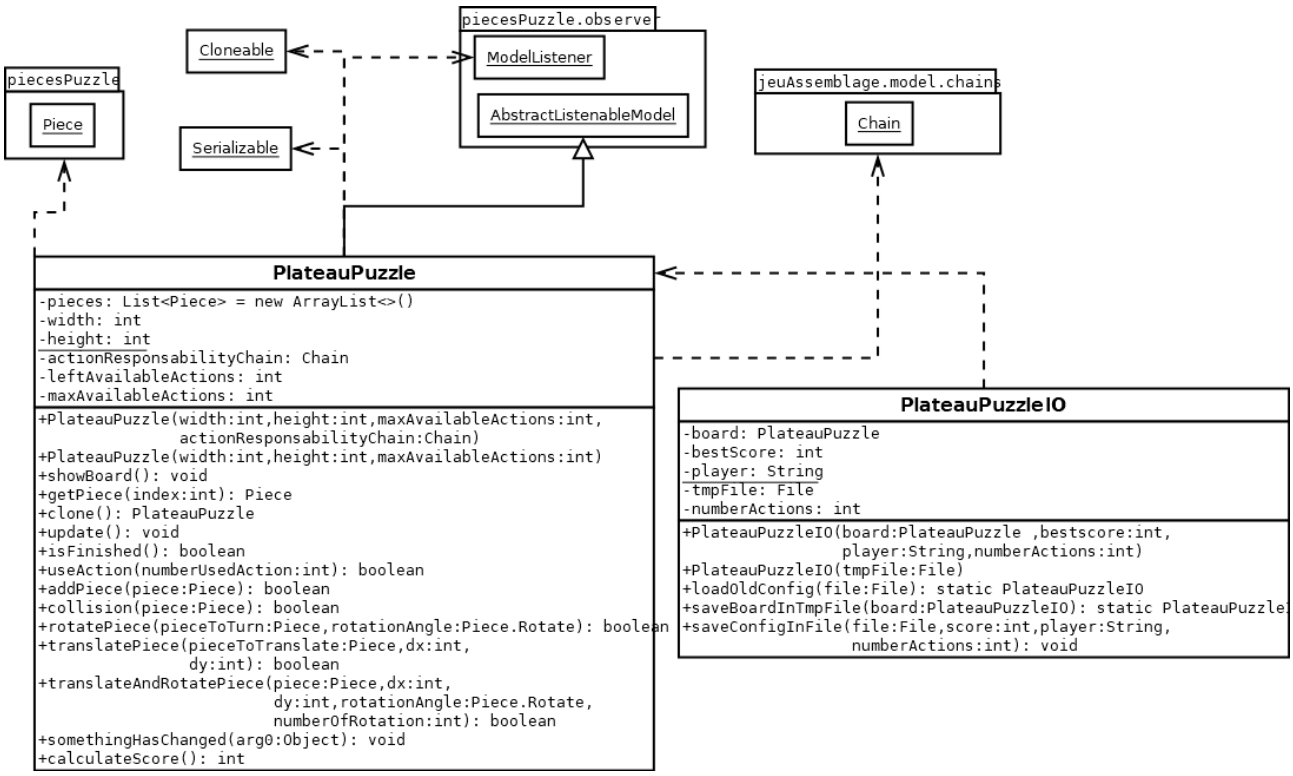
Voici la structure globale de notre package ‘jeuAssemblage’. Nous avons choisi de subdiviser le package ‘jeuAssemblage.model’ car selon nous, il était plus approprié de ne pas garder toutes les classes à un seul niveau de package. Pour la division des sous-packages, nous avons juste séparé en fonction des pattern implémentés comme vous le verrez dans les diagrammes suivants.

Voici le diagramme de classes pour le package ‘jeuAssemblage.model.chains’ :



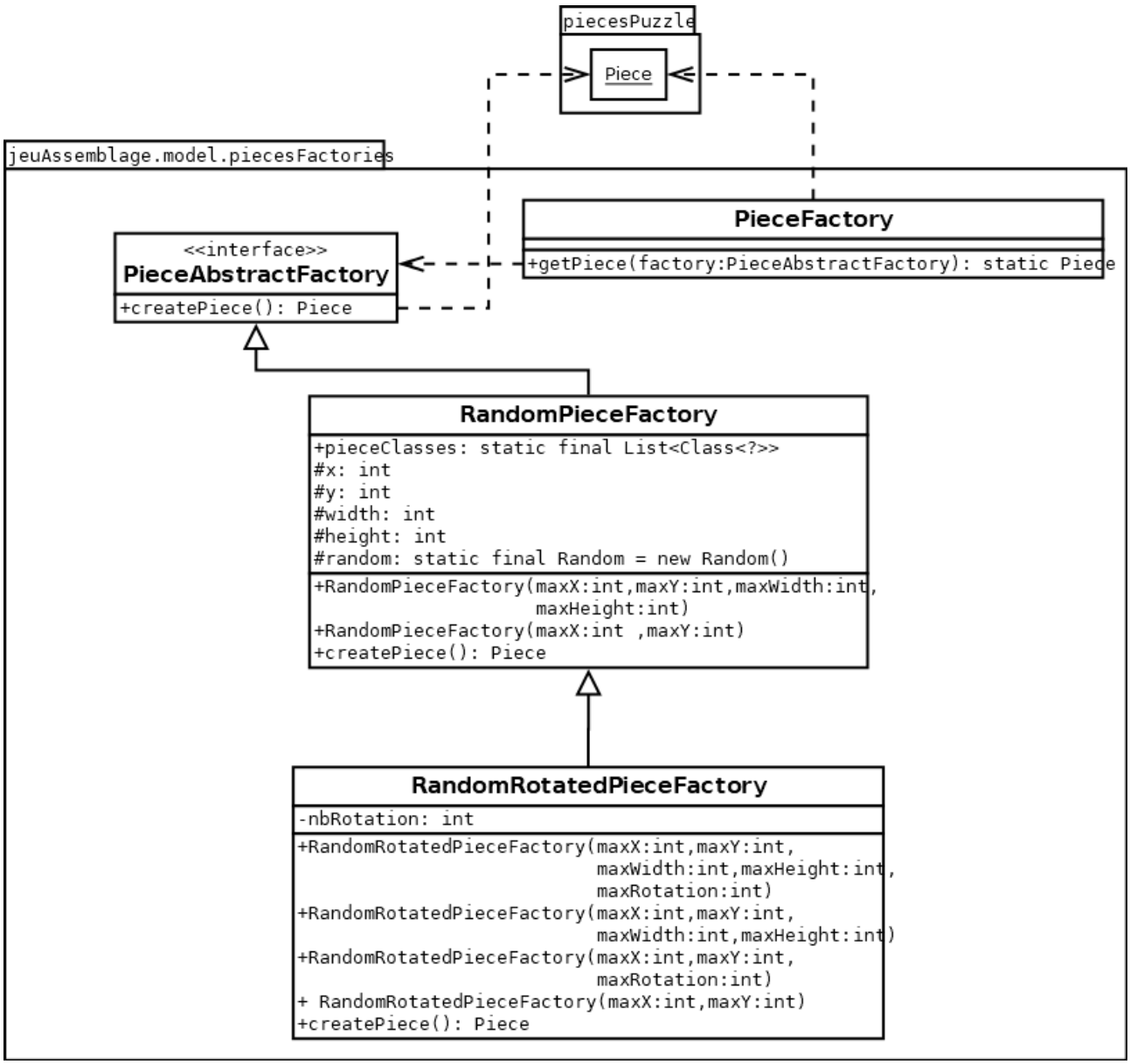
Ce sous-package contient des classes permettant de faire les vérifications lors d’un ajout ou d’une modification d’une pièce dans le plateau de jeu. Pour cela, on utilise le pattern Chain of Responsability, permettant de tester les éléments du plateau grâce àl’exécution des maillons de la chaîne les uns à la suite des autres. Cela permet aussi de rajouter simplement un maillon en bout de chaîne afin d’ajouter une nouvelle vérification sur une pièce. Cette structure permet donc de rajouter des vérifications sans avoir à modifier le code du plateau de code et cela de façon dynamique.

Voici le diagramme de classes pour le package ‘jeuAssemblage.model’ sans ses sous-packages :



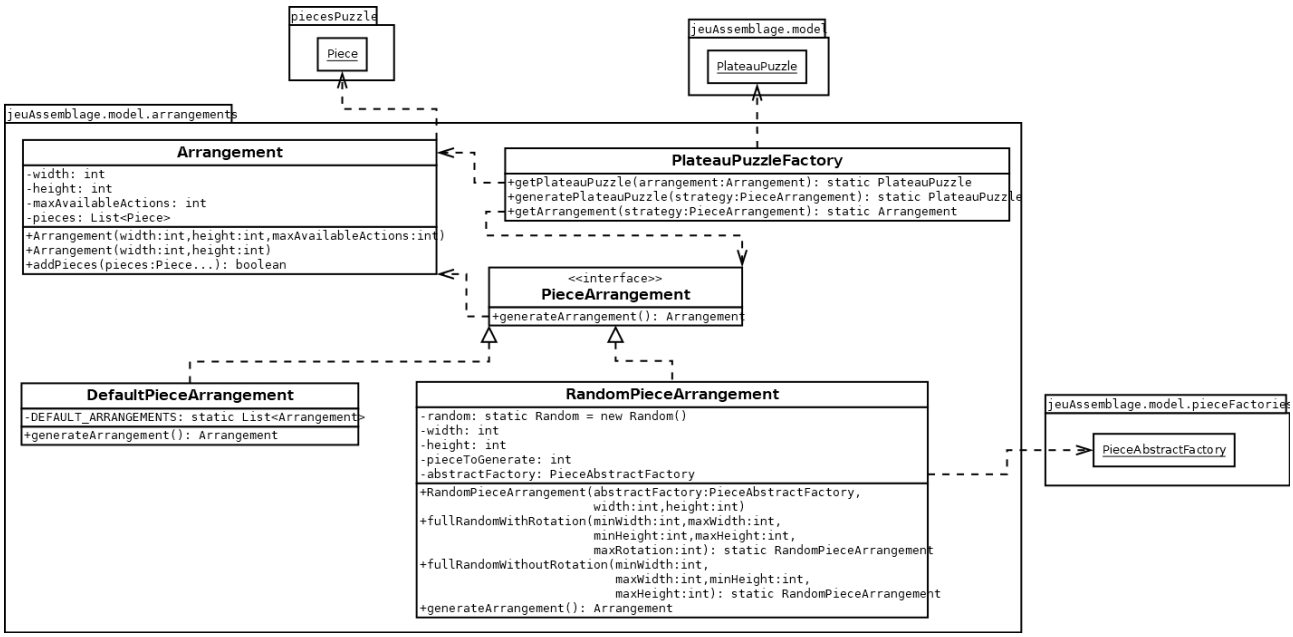
Ici, nous avons notre classe modèle qui est ‘PlateauPuzzle’. Nous pouvons ajouter des pièces, les déplacer, les tourner, et effectuer ses 2 actions en une seule fois. Cette classe contrôle les pièces grâce à la chaîne de responsabilité dont le diagramme a été montré précédemment et modifie le nombre de coups restants en fonction des actions effectuées et de leurs paramètres (optimisation des coups, ex : on ne compte pas un coup en moins si le joueur déplace la pièce avec un vecteur (0,0)). De l’autre côté, nous avons la classe ‘PlateauPuzzleIO’, qui permet de sauvegarder la configuration d’un plateau et de la recharger en mémoire.

Voici le diagramme de classes pour le package ‘jeuAssemblage.model.pieceFactories’ :

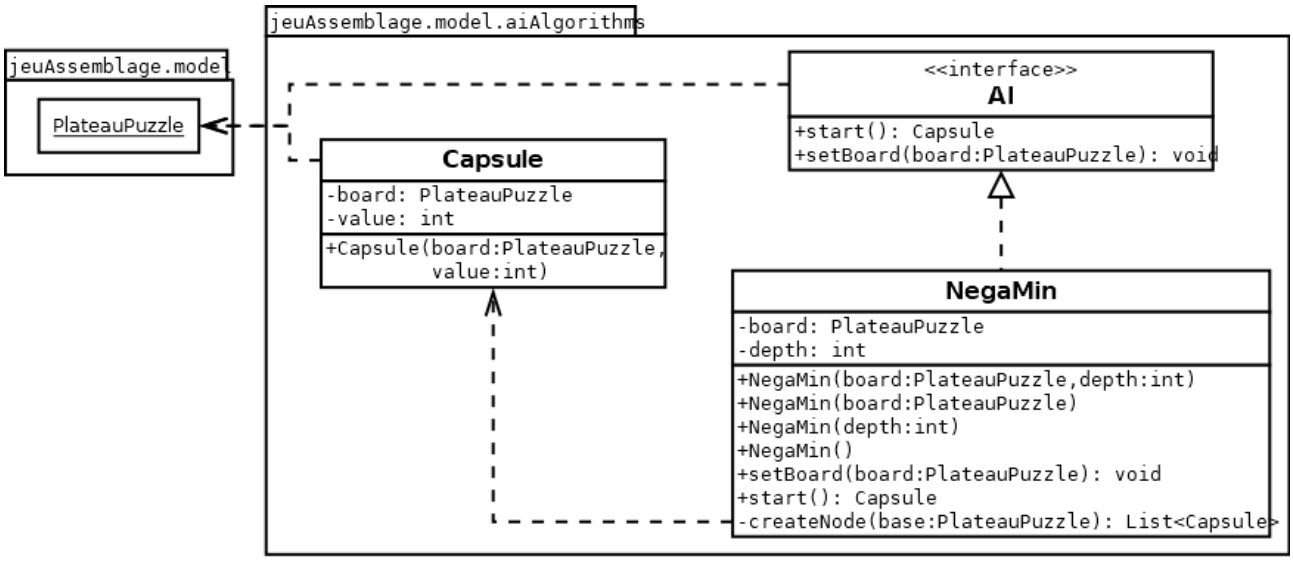


Ce sous-package nous permet de générer des pièces permettant ensuite, de les ajouter sur un plateau de jeu avec l'usine choisie. Nous utilisons le pattern Strategy ici afin de pouvoir changer aisément les générateurs de pièces et pouvoir en ajouter d'autres. Nous pourrions aussi bien ajouter des usines suivant tel système de génération de pièce lié à une formule mathématique ou non, à une certaine logique, etc.

Voici le diagramme de classes pour le package 'jeuAssemblage.model.arrangements' :



Ce sous-package nous permet de générer des arrangements de pièces permettant ensuite, de générer un plateau de jeu avec l'arrangement généré (ou choisi). Nous utilisons le pattern Strategy ici afin de pouvoir changer aisément les générateurs d'arrangements et pouvoir en ajouter d'autres suivant d'autres systèmes de génération.



Voici le diagramme de classes pour le package 'jeuAssemblage.model.aiAlgorithms' :
Dans ce sous-package, on utilise le pattern Strategy afin de laisser le choix à l'utilisateur entre les différents algorithmes d'intelligence artificielle implémentées dans celui-ci. Ici, nous n'avons implémenté qu'un seul algorithme mais grâce à ce pattern, nous pourrions en rajouter. Pour plus de détails sur l'algorithme, veuillez lire la partie suivante.

Explication des algorithmes non triviaux implémentés :

1. L'intelligence artificielle : nous avons décidé d'implémenter une variante de l'algorithme MinMax, qui permet d'effectuer des actions tout en minisant la perte maximum dans un jeu. Dans le nôtre, il n'y a qu'un joueur et le score doit être minimisé. Nous avons donc adpater l'algorithme pour faire une recherche afin de seulement minimiser le score obtenu (but du jeu).
   Nous avons implémenter la variante simplifiée NégaMax (ou plutôt NégaMin dans notre cas). À chaque nœud, nous recherchons toutes les actions possibles pour toutes les pièces et les effectuons tout en clonant le plateau lors de la réalisation de celles-ci uniquement si

celle-ci est inférieure ou égale au score de l'état courant du plateau. L'algorithme ensuite, effectue récursivement une recherche sur les plateaux ayant effectués l'action. Il retourne en arrière seulement lorsque le jeu est soit terminé ou que la profondeur de raisonnement de l'algorithme est atteinte.

2. Rotation des pièces : pour l'implémentation des pièces, afin de ne pas utiliser trop d'espace mémoire, nous avons décidé d'implémenter un algorithme de rotation qui tourne directement les case de la pièce et modifie donc la longueur et la hauteur de la pièce de manière automatique. Cet algorithme n'utilise certes pas la mémoire, mais utilise la capacité de calcul du processeur.