Université de Caen Normandie

Méthode de Conception :

Rapport : Jeu d'assemblage de formes

Professeur :
M. Yann Mathet

Groupe
ADOUMA HASSAN 2B
MAHUDO KOTIN 2B
ESSAM DONNEL 3B
SOULEYMANE BARY 2A

Licence 3 Informatique : 5 DECEMBRE 2023

Table des matières

0.1 Introduction :

En licence 3 informatique à l'université de Caen Normandie, la méthode de conception est une discipline essentielle pour une compréhension approfondie de la programmation orientée objet. Cette approche permet d'acquérir des principes fondamentaux tels que la modularité et la maintenabilité, ainsi que la mise en pratique des design patterns.

Le développement logiciel, en constante croissance, nécessite des compétences spécifiques pour relever les défis complexes qui émergent. Afin d'assurer la qualité des logiciels, l'application d'architectures de développement et de patrons de conception est devenue cruciale. Ces approches structurées permettent de résoudre efficacement les problèmes récurrents, favorisant la maintenabilité et la réutilisabilité du code. Dans le cadre d'un exemple concret, tel qu'une application de jeu d'assemblage de formes, l'utilisation d'une architecture MVC et de patrons comme le patron stratégie simplifie le développement et offre une flexibilité essentielle. Ainsi, intégrer ces pratiques est impératif pour tout développeur moderne afin de garantir le succès des projets logiciels dans un environnement en constante évolution. C'est dans ce contexte que nous nous sommes réunis par groupe de 4 étudiants pour atteindre cet objectif.

0.2 Principe du Jeu d'assemblage des formes :

Le jeu d'assemblage de formes vise à créer une application dotée d'une interface graphique où le joueur doit assembler différentes formes de manière à occuper le moins d'espace possible sur l'aire de jeu. Inspiré du célèbre Tetris, ce jeu présente des modalités distinctes, telles que l'exposition de toutes les pièces dès le début de la partie, la possibilité pour le joueur de choisir une pièce à tout moment, et l'objectif de minimiser l'espace occupé par l'ensemble des pièces.
Le joueur peut faire tourner ou déplacer les pièces pour optimiser l'occupation de l'espace. L'évaluation se fait en fonction de l'aire du plus petit rectangle (parallèle aux axes) recouvrant l'ensemble des pièces. Lorsque le joueur estime avoir terminé ou atteint le nombre maximum d'actions autorisées, son score est calculé
Le jeu propose également une option permettant à l'ordinateur de jouer, avec la possibilité de mettre en place des stratégies pour le joueur robot, définies par le design pattern Strategy. Différents types de formes peuvent être générés de manière aléatoire, offrant une variété de configurations au jeu
La conception du jeu intègre des design patterns, notamment Strategy pour les stratégies du joueur robot, Chain of Responsibility pour vérifier la validité des actions du joueur, et Factory pour la création des pièces lors d'une nouvelle partie. L'application adopte une architecture MVC complète, permettant une jouabilité aussi bien avec une interface graphique qu'en ligne de commande sans modification du modèle.

0.3 Présentation du projet :

Le projet consiste à la réalisation de deux applications.
L'une étant une application Jeu d'assemblage utilisant la librairie proposée par piecesPuzzle.jar

0.3.1 Application JeuxAssemblageMVC :

L'application en question, dont la structure est illustrée dans le schéma ci-dessous :

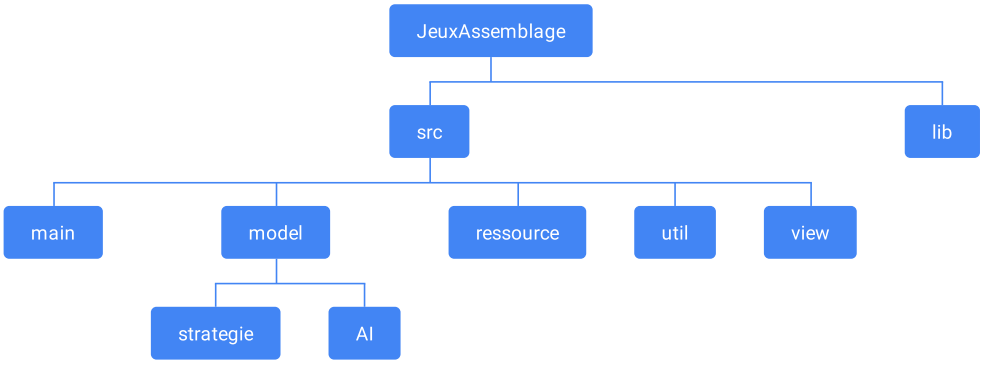

FIGURE 1 – Arborescence du dossier jeuAssemblage :

0.3.2 L'application PiecePuzzle :

L'arborescence de l'application piecePuzzle

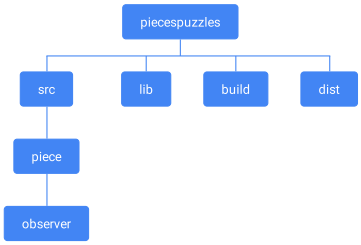

FIGURE 2 – Arborescence de L'application PiecePuzzle :

0.4 Organisations du M-VC(Modèle - VueControleur) :

Dans le projet, le pattern Observer est utilisé pour mettre en œuvre le modèle MVC (Modèle-Vue-Contrôleur). Pour le plateau de jeu par exemple, le modèle est représenté par la classe Plateau, qui est responsable de la logique métier concernant le plateau de jeu. La vue est représentée par la classe BoardView, qui est chargée de l'affichage graphique du plateau de jeu. Le pattern Observer permet à la vue d'être informée des changements dans le modèle sans avoir à connaître les détails internes du modèle. La classe abstraite héritée par le modèle Plateau AbstractListenableModel agit en tant qu'observable et BoardView implémente l'interface ListeningModel, agissant en tant qu'observateur. Lorsque des changements surviennent dans le modèle, tels que l'ajout ou la suppression de pièces sur le plateau, la méthode fireChangement hérité de AbstractListenableModel est appelée, notifiant ainsi tous les observateurs enregistrés, comme la vue BoardView, de la mise à jour. La méthode modelUpdate de BoardView est appelée lorsqu'un changement est détecté dans le modèle (Plateau). Dans cette méthode, la vue réagit en conséquence en mettant à jour l'affichage du plateau de jeu. Plus précisément, elle réinitialise l'affichage, puis ajoute les nouvelles pièces en fonction de l'état actuel du modèle

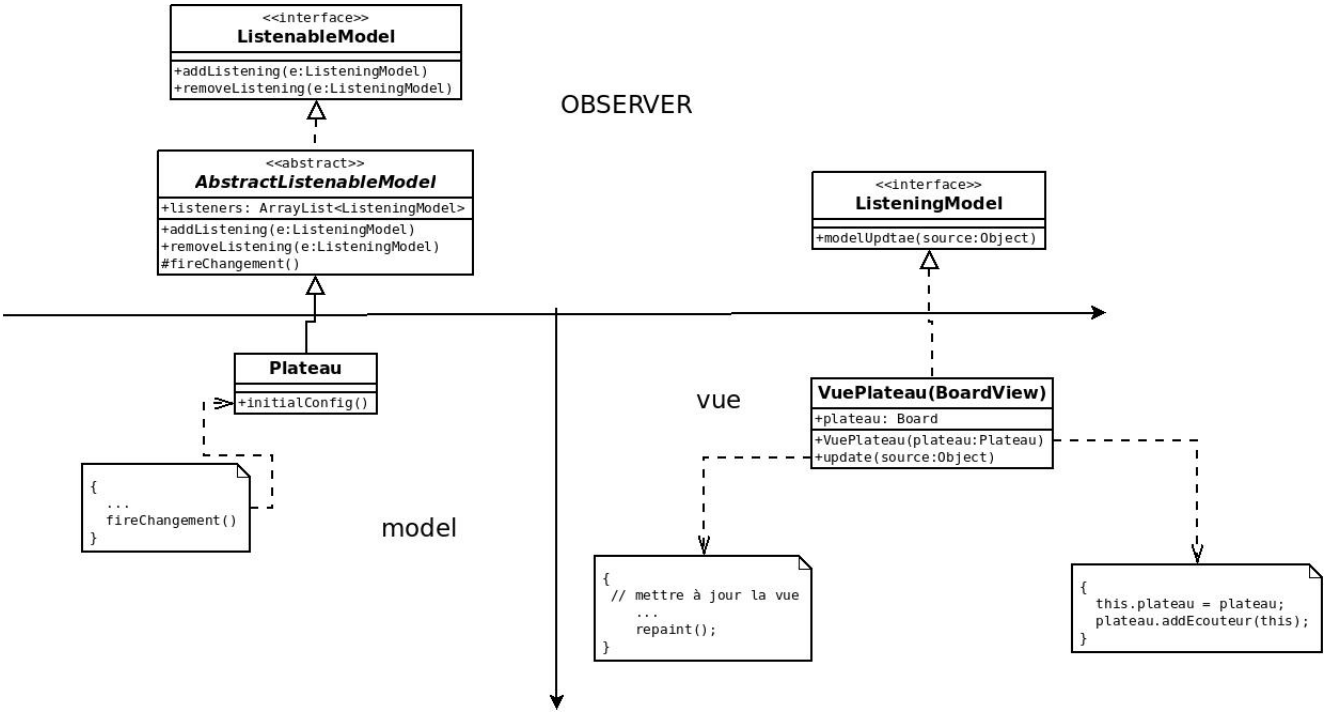

FIGURE 3 – diagramme du M-VC :

0.5 Utilisation des design Patterns

0.5.1 Le pattern Strategy :

Le pattern Strategy est utilisé dans ce code pour définir différentes stratégies de placement des pièces sur un plateau de jeu. Deux classes, ArbitraryStrategy et RandomStrategy, implémentent l'interface PlacementStrategy, chacune définissant une méthode initialConfig permettant de configurer le plateau selon une stratégie spécifique. La classe ArbitraryStrategy prend en compte une configuration prédéfinie avec des coordonnées spécifiques pour chaque pièce, offrant ainsi une approche statique et déterministe du placement. En revanche, la classe RandomStrategy utilise une approche dynamique en générant aléatoirement des coordonnées pour chaque pièce jusqu'à ce que le nombre requis de pièces soit placé sur le plateau. La classe principale Plateau utilise une instance de l'interface PlacementStrategy pour déléguer la responsabilité de la configuration initiale du plateau. Cela permet de facilement changer la stratégie de placement à l'exécution en affectant une nouvelle instance de stratégie à l'attribut placementStrategy. Ainsi, le pattern Strategy offre une flexibilité et une extensibilité au système, permettant de facilement ajouter de nouvelles stratégies de placement sans modifier le code existant de la classe Plateau.

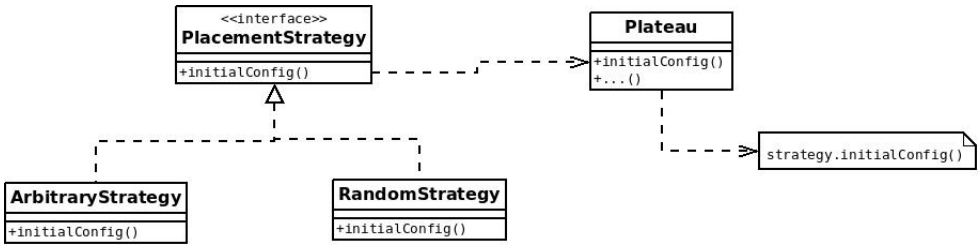

FIGURE 4 – diagramme du pattern strategy :

0.5.2 Le pattern factory :

Pour la création des pièces nous avons utilisé **le pattern factory.** En utilisant une classe dédiée, **la PieceFactory**, pour la création des instances de pièces, nous avons encapsuler les détails de construction, permettant ainsi aux **clients (classes du jeu assemblage)**d'obtenir des objets pièce sans se soucier des détails spécifiques de chaque type de pièce ou de la logique de construction. De plus, nous utilisons des méthodes de configuration fluides qui offrent une approche élégante pour personnaliser les attributs des pièces lors de leur création, facilitant ainsi la création d'objets avec différentes configurations. Ainsi notre factory nous permet de créer des pièces de manière aléatoire ou avec une configuration défini . En centralisant la logique de création dans la classe PieceFactory, le code devient plus lisible, modulaire et extensible, permettant l'ajout aisé de nouveaux types de pièces à l'avenir sans perturber le reste du système.

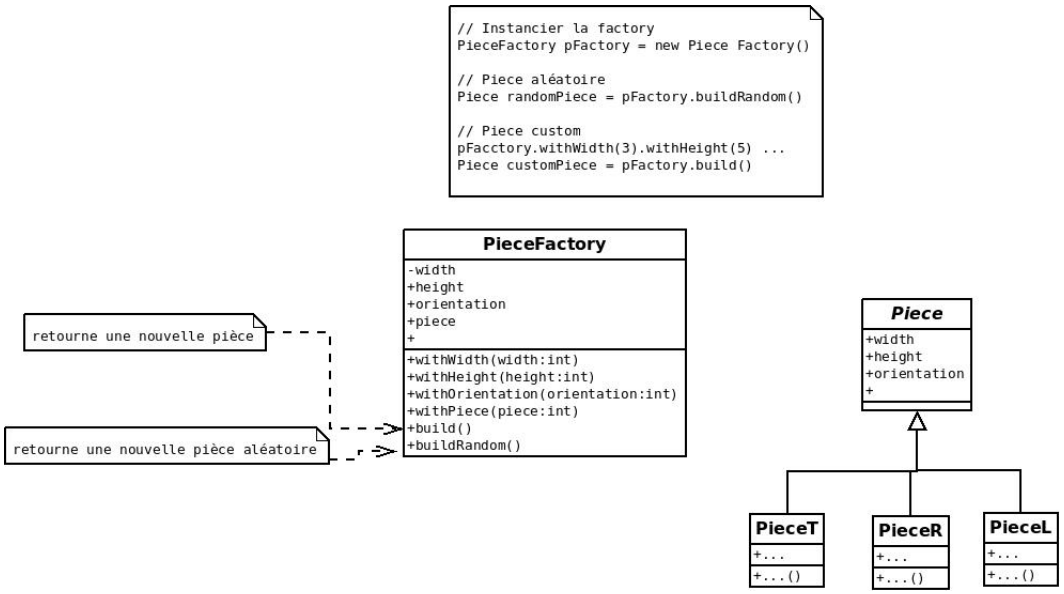

FIGURE 5 – diagramme du pattern factory :

0.5.3 Chain of Responsibility

Le pattern Chain of Responsibility est utilisé dans le projet pour gérer la validation du placement des pièces sur le plateau de jeu. La classe abstraite AbstractValidator définit une chaîne de validateurs, où chaque validateur a la possibilité de traiter la demande de validation ou de la transmettre au validateur suivant dans la chaîne.Dans ce contexte, deux validateurs concrets, OverflowValidator et OverlappingValidator, sont implémentés. OverflowValidator vérifie si la pièce déborde du plateau, tandis que OverlappingValidator s'assure qu'il n'y a pas de chevauchement entre les pièces déjà placées et la nouvelle pièce. La méthode statique buildChain de la classe abstraite permet de construire facilement une chaîne de validateurs en spécifiant le premier validateur et les validateurs suivants. Ainsi, la chaîne de validation est créée avec OverflowValidator en premier, suivi de OverlappingValidator. Lorsqu'une demande de validation est faite à la chaîne, le premier validateur (ici OverflowValidator) traite la demande et, s'il le souhaite, la transmet au validateur suivant (OverlappingValidator). Chaque validateur décide s'il peut valider la demande ou s'il doit la transmettre au validateur suivant. Cela offre une flexibilité et une extensibilité dans l'ajout de nouveaux validateurs sans modifier le code existant, facilitant ainsi la maintenance et l'évolution du système de validation du placement des pièces.

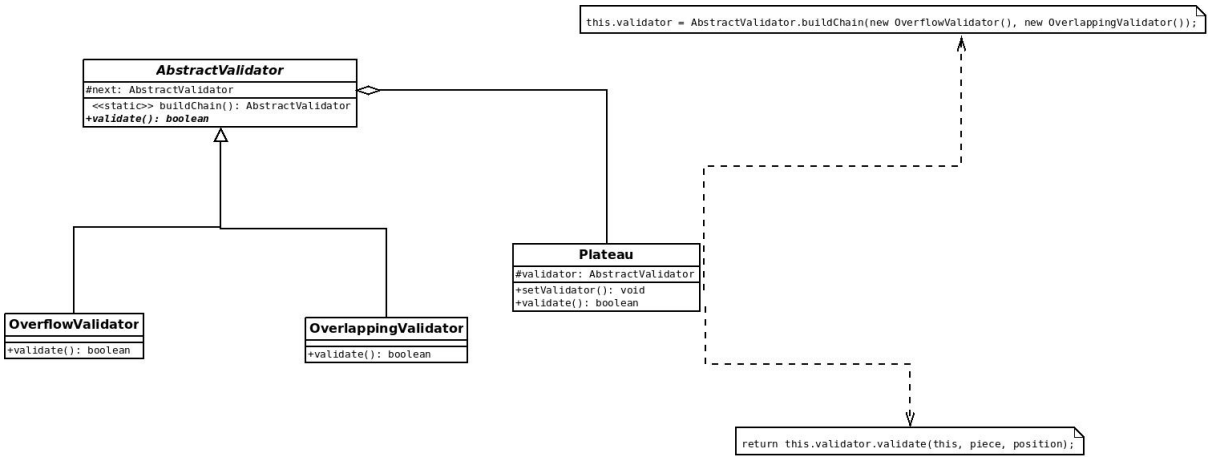

FIGURE 6 – diagramme du pattern chaine of responsibility :

0.6 Conclusion

La réalisation de ce projet a constitué une opportunité précieuse pour approfondir notre compréhension des concepts abordés initialement. En mettant en pratique les connaissances acquises, nous avons consolidé notre maîtrise des principes de conception logicielle, notamment l'utilisation des design patterns et l'application de l'architecture MVC.

Ce projet a été une occasion concrète de consolider nos compétences en programmation orientée objet, en stratégies de conception, et en résolution de problèmes complexes inhérents au développement logiciel. En synthèse, la mise en œuvre de ce projet a enrichi notre bagage de compétences et renforcé notre capacité à aborder des défis concrets dans le domaine de l'informatique