Louis SIMIER
ING1 GI Groupe 3

Rapport Projet Java

Résolution de puzzles

2024/2025

Sommaire

Problèmes et Solutions :

Au cours de mon avancée sur ce projet, j’ai été confronté à de nombreux problèmes techniques et méthodologiques. Ce projet m’a tout d’abord permis de découvrir des outils que je n’avais pas encore utilisés pleinement, comme Git et GitHub. Lors de mes premières manipulations, j’ai rencontré des difficultés à cloner un dépôt GitHub dans Visual Studio Code. Après plusieurs tentatives infructueuses, j’ai consulté des forums, ce qui m’a permis d’identifier une solution.

Une fois ce premier obstacle franchi, j’ai commencé à créer la structure de base de mon projet Java et à écrire du code. Rapidement, de nouveaux défis sont apparus. Le traitement d’images de pièces de puzzle s’est révélé complexe : il fallait détecter les bords, distinguer les formes et organiser les pièces de manière cohérente, sans orientation initiale. Les comparateurs de forme que j’avais initialement codés ne donnaient pas des résultats fiables. J’ai alors dû reconsidérer ma méthode d’analyse des images.

Dans un premier temps, j’ai tenté de comparer les pixels en analysant directement leurs valeurs RGB (rouge, vert, bleu). J’avais vu cette idée sur un forum qui en parlait : l’idée était de détecter la continuité ou la rupture entre les bords de deux pièces. Cependant, cette approche s’est révélée peu efficace, car des variations mineures d’éclairage ou de couleur généraient de fausses incompatibilités. De plus, j’avais l’impression que certaines pièces se séparaient à cause de différences mineures dans les pixels de bord. Pour atténuer ce problème, j’ai essayé de convertir les pixels en niveaux de gris afin de simplifier l’analyse en réduisant l’information à un seul canal d’intensité lumineuse. Malgré ces ajustements, les résultats restaient difficiles à interpréter.

J’ai également tenté de développer une méthode basée sur la reconnaissance du contour global des pièces, dans l’espoir de les positionner en fonction de leur forme, comme on le ferait pour reconstituer le cadre d’un puzzle en plaçant d’abord les bords. L’idée était d’analyser les pixels extérieurs pour détecter des motifs réguliers ou plats (indiquant un bord) et d’en déduire leur position (haut, bas, gauche, droite). Cependant, cette approche s’est révélée incomplète et difficile à généraliser : certaines pièces avaient des formes trop complexes ou trop proches les unes des autres, et l’analyse du contour ne permettait pas de tirer des conclusions fiables.

Un de mes plus gros problèmes a été le temps, car j’en ai perdu beaucoup à vouloir créer quelque chose d’universel qui fonctionnerait pour n’importe quel puzzle, aussi complexe soit-il. Or, j’ai vite été rattrapé par le fait que je code en Java depuis moins de cinq mois, et que les mécanismes nécessaires me semblaient trop avancés et trop complexes, tant dans l’utilisation que dans la compréhension.

Pour obtenir au final quelque chose qui fonctionne, je suis revenu à mon idée de départ, qui ne me semblait pas optimale, mais plus pratique. J'ai développé une méthode de conversion des images en matrices binaires représentant la présence ou non de contenu visuel (1 pour les pixels "non vides", 0 sinon). Ensuite, j'ai mis en place un comparateur ligne par ligne basé sur la distance entre les bords des pièces, en affinant progressivement la gestion des cas de chevauchement ou de trous entre les pièces. L'idée était que si j'arrivais à définir le dernier pixel à droite d'une pièce et le premier à gauche d'une autre, je pourrais détecter s'il s'agissait d'un bon appariement.

D'autres problèmes incluaient la détection de l'ordre des pièces sans point de départ fixe, la construction automatique de lignes horizontales, ou encore l'estimation de la hauteur réelle des pièces malgré les irrégularités dues aux formes du puzzle. Ces obstacles ont été résolus par des ajustements progressifs du code et des phases de test fréquentes, avec des impressions de matrices, des vérifications visuelles, et la création d'outils de débogage pour interpréter les décisions de l'algorithme. J'ai donc passé beaucoup de temps à créer quelque chose, le perfectionner, l'analyser à base de « println », pour finalement l'abandonner au profit d'une solution plus simple.

Enfin, la gestion des fichiers, la génération automatique des images de la solution, et la structuration du code en classes cohérentes ont aussi demandé du temps. Mais au final, toutes ces difficultés m'ont permis de renforcer mes compétences en programmation Java, en traitement d'images simples, et en gestion de projet avec GitHub.

Limites fonctionnelles

Malgré les nombreuses fonctionnalités développées et les solutions mises en place, le projet présente encore certaines limites fonctionnelles.

Tout d'abord, le système de comparaison des pièces repose sur des heuristiques simples, principalement basées sur l'analyse binaire de la forme (présence ou absence de pixels). Cette méthode ne prend pas en compte les subtilités des formes, comme les courbes douces. De plus, elle reste sensible aux irrégularités sur les bords des pièces, dues à des pixels parasites ou à une mauvaise découpe dans les images d'origine.

Une autre limite importante concerne la méthode de conversion des images en matrices binaires. Actuellement, le système considère qu'un pixel est « vide = 0 » uniquement s'il est strictement noir (RGB = 0,0,0). Or, cela peut poser problème si la pièce de puzzle elle-même contient du noir dans son design. Par exemple, on pourrait avoir un motif ou une teinte sombre sur la pièce. Dans ce cas, le programme peut interpréter ces pixels comme du vide alors que ce n'est pas le cas, ce qui perturbe la comparaison des bords ou le calcul de la hauteur « réelle ».

Des méthodes plus avancées, comme la détection de contours, permettraient de distinguer plus finement ce qui est réellement la bordure de la pièce par rapport à un simple noir présent dans le motif. Néanmoins, ces approches sont plus complexes à implémenter et nécessitent un traitement d'image plus poussé.

Pour le moment, afin de conserver une méthode simple et robuste, le programme autorise une certaine permissivité : des erreurs localisées (pixels noirs interprétés à tort) sont tolérées, à condition qu'elles ne se reproduisent pas de manière excessive ou systématique. Cela permet au comparateur de rester fonctionnel, même avec des imperfections dans les images sources.

Ensuite, l'algorithme de construction des lignes suppose que toutes les pièces sont parfaitement alignables dans une ligne horizontale. Il ne gère pas les cas où une pièce pourrait appartenir à une ligne supérieure ou inférieure. En particulier, il n'y a pas de vérification géométrique globale une fois la ligne construite. En effet, pour l'instant, le code identifie les pièces par leur taille « réelle ». Il serait problématique que toutes les pièces aient la même taille « réelle » mais ne soient pas destinées à la même ligne.

De plus, le système ne permet pas encore d'assembler toutes les lignes entre elles pour reconstituer un puzzle complet. Les lignes sont bien construites individuellement, mais leur agencement vertical reste à améliorer.

Enfin, l'interface est inexistante : le projet ne dispose pas d'une interface graphique interactive, et tout le traitement repose sur des logs console et la génération d'images de sortie. Cela rend l'utilisation moins intuitive et impose une bonne connaissance du fonctionnement interne pour interpréter les résultats.

Malgré ces limites, le projet constitue une base solide, et beaucoup de ces aspects pourraient être améliorés dans une version ultérieure : gestion avancée des formes, détection automatique de la structure du puzzle, ou encore intégration d'une interface utilisateur.

Planning de réalisation

Mercredi 14 Mai : Création du GitHub, du projet sur VSCode et création de la structure du projet.

Jeudi 15 et Vendredi 16 Mai : Début du code et recherche sur la manière de gérer et utiliser des images.

Week-end du 17 Mai : Finalisation du code de base.

Lundi 19, Mardi 20 et Mercredi 21 Mai : Début de recherche, code et test multiple sur la manière de comparer des pièces de puzzles via la manière RGB mais aussi via les formes.

Jeudi 22, Vendredi 23 et Samedi 24 Mai : Abandon de beaucoup de code de comparaison, écriture et réalisation du code final de comparaison ligne par ligne via les matrices binaires.

Dimanche 25 Mai : Rapport de fin de projet.

Amélioration Possibles

Plusieurs pistes d'amélioration sont envisageables pour faire évoluer ce projet vers une solution plus complète et plus précise. Tout d'abord, l'algorithme de comparaison pourrait être renforcé afin d'identifier des pièces parfaitement compatibles, même si elles sont orientées différemment. Actuellement, les comparaisons se font uniquement sans rotation ; or, dans un puzzle réel, une pièce peut s'imbriquer à condition d'être tournée. Il serait donc pertinent d'implémenter une version de la comparaison qui teste toutes les rotations possibles (90°, 180°, 270°), afin de garantir que l'emboîtement soit détecté indépendamment de l'orientation initiale.

De plus, dans le cas où plusieurs pièces présentent une forme très similaire ou quasiment identique, il devient difficile de trancher avec certitude sur leur compatibilité uniquement à partir de la forme binaire. Pour résoudre ce problème, il serait possible de réintroduire une analyse basée sur les couleurs RGB, comme celle envisagée au début du projet. Cette approche permettrait de départager deux pièces qui ont des contours identiques, mais des motifs ou des teintes différentes. Elle pourrait ainsi agir en complément du comparateur de forme, en particulier dans les cas ambigus.

Enfin, des outils de visualisation plus poussés et une interface utilisateur permettraient de mieux interpréter les décisions prises par l'algorithme, et faciliteraient les ajustements futurs.

Diagramme de classes

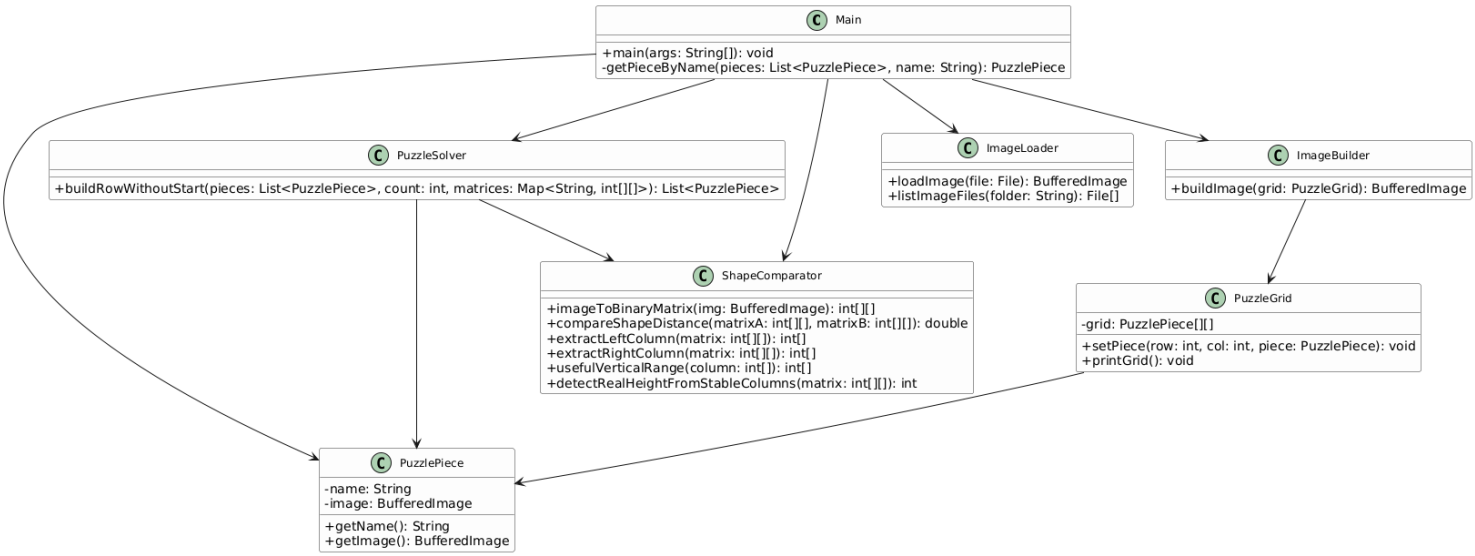
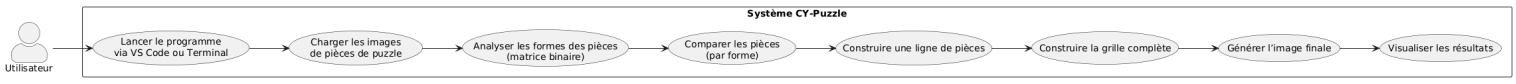


Diagramme des cas d’utilisations



Javadoc

```
<!DOCTYPE HTML>
<html lang="fr">
<head>
<!-- Generated by javadoc (17) on Mon May 25 16:07:22 CEST 2025 -->
<title>Generated Documentation (Untitled)</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<meta name="dc.created" content="2025-05-26">
<meta name="description" content="index redirect">
<meta name="generator" content="javadoc/IndexRedirectWriter">
<link rel="canonical" href="puzzle/package-summary.html">
<link rel="stylesheet" type="text/css" href="stylesheet.css" title="Style">
<script type="text/javascript">window.location.replace('puzzle/package-
summary.html')</script>
<noscript>
<meta http-equiv="Refresh" content="0;puzzle/package-summary.html">
</noscript>
</head>
<body class="index-redirect-page">
<main role="main">
<noscript>
<p>JavaScript is disabled on your browser.</p>
</noscript>
<p><a href="puzzle/package-summary.html">puzzle/package-summary.html</a></p>
</main>
</body>
</html>
```

Conclusion

Ce projet Java m’a permis de consolider mes compétences en programmation orientée objet, en manipulation d’images et en algorithmique appliquée à un cas concret : l’assemblage automatique d’un puzzle numérique. Au fil du développement, j’ai appris à structurer un projet en plusieurs classes cohérentes, à utiliser des outils comme Git, Visual Studio Code et Javadoc, ainsi qu’à surmonter des difficultés techniques telles que la comparaison de formes ou la gestion des bords des pièces. Bien que certaines approches, comme la tentative de reconstruction par contours, se soient révélées incomplètes, elles ont contribué à enrichir ma compréhension des limites possibles. Ce travail réalisé tous seul ma finalement permis de voir et de m’améliorer dans beaucoup de points.