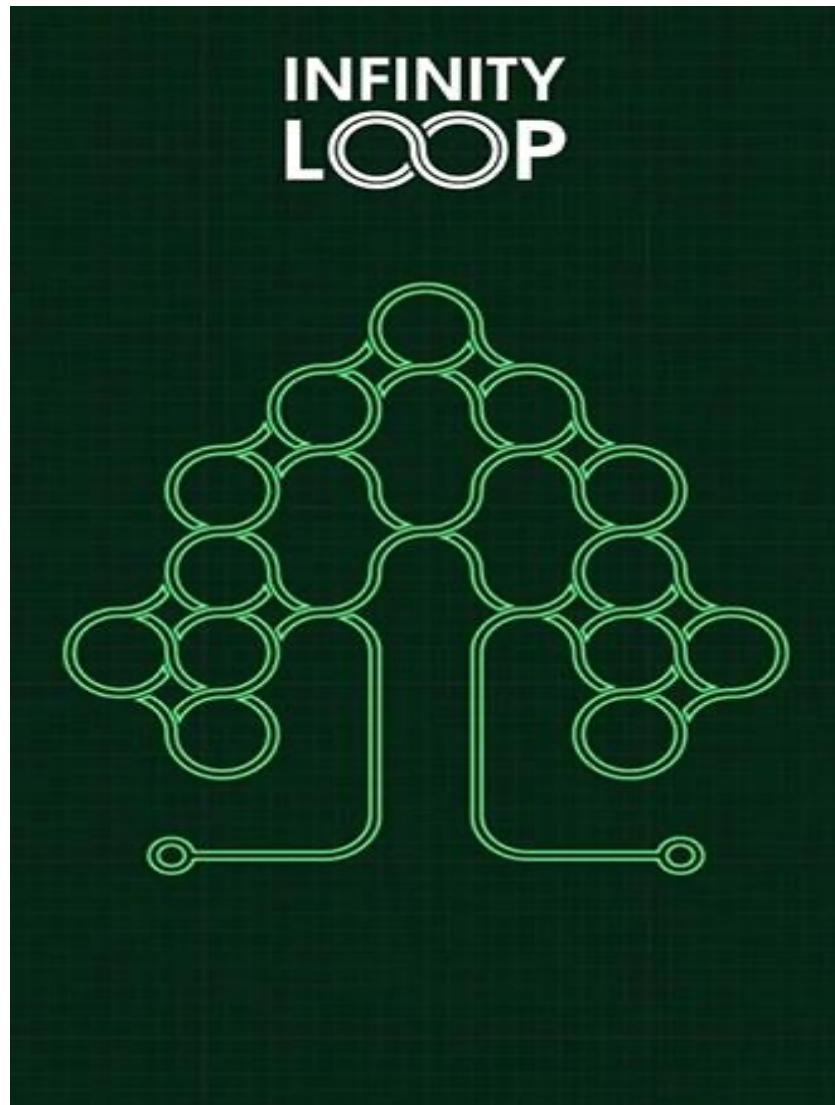


Rapport de projet

Projet de programmation Orientée Objet Java

UE M1 MIAGE - Hossein Khani



Farès AMIAR
Rayan PREVOST
Youssef TEKAYA

M1 MIAGE 2021 - 2022

Sommaire

1 - Présentation rapide du projet et de notre équipe de projet	2
2 - Présentation de notre démarche	3
3 - Présentation de notre réalisation et de la réflexion de notre solution	4
Generator	4
Solver & Checker	5
GUI	7
4 - Présentation des résultats et ressenti sur le projet	8

I - Présentation rapide du projet et de notre équipe de projet

Pour le cours de Programmation Orientée Objet Avancée en JAVA, nous avons eu la possibilité de travailler sur un problème assez complexe visant à travailler autour du célèbre jeu Infinity Loop.

Pour présenter rapidement, dans ce jeu, un niveau est constitué d'une grille où chaque case (cellule) peut contenir (ou non) une pièce. Le but du jeu est de relier l'ensemble des pièces. Dans notre projet, nous supposons que chaque cellule peut être vide ou qu'elle peut prendre un pièce en fonction de figures bien précises visibles ci-dessous.



Un niveau est considéré comme résolu, si tous les composants sont reliés. Notre but était de mettre en place un générateur de niveau, un checker de solution, un résolveur de niveau et un interface graphique permettant de visualiser le niveau du jeu.

2 - Présentation de notre démarche

Avant de débuter la partie codage, nous avons pris le temps de bien comprendre les fichiers fournis ainsi que le sujet et ce qui nous était demandé de mettre en place. C'est l'étape la plus importante dans le début d'un projet car elle nous permet d'être sûr que nous avons bien pris connaissance de l'ensemble des fonctions déjà mise en place dans l'optique de ne pas réinventer la roue.

Initialement nous voulions utiliser la méthode TDD pour avancer sur la partie codage de notre projet, mais nous avons connu de nombreux soucis de planning et d'organisations, que nous expliquerons plus tard, ne nous permettant d'utiliser cette technique de programmation. Nous avons donc décidé de diviser le projet entre nous afin que chaque personne travaille principalement sur une partie, nous permettant d'avancer plus rapidement sur les fonctions et les fonctionnalités les plus utiles pour le projet.

Pour pouvoir collaborer entre nous, nous avons décidé d'utiliser GIT comme outils de versionning de notre code. En effet, nous avons essayé d'effectuer des commits assez régulièrement pour pouvoir être sûr de savoir à chaque instant T ce sur quoi l'un et l'autre travaille, dans le but de ne pas empiéter sur sa conception.

En parallèle, nous avons organisé des réunions via discord le plus souvent possible pour que l'on puisse confronter nos idées et notre compréhension du sujet, mais aussi et surtout afin de pouvoir échanger sur nos difficultés, et réfléchir ensemble sur les meilleures choses à mettre en place afin d'avancer de manière optimale dans notre développement.

3 - Présentation de notre réalisation et de la réflexion de notre solution

Afin d'avancer de manière optimale, nous avons tout d'abord décidé de diviser le travail permettant à chaque membre du projet de pouvoir coder ces fonctionnalités et de rapporter sa pierre à l'édifice.

Tout d'abord, nous avons dû convertir le dossier en un dossier Maven, un des problèmes que nous avons rencontré était le fait que nous n'utilisons pas forcément la même version du JDK de Java, provoquant de nombreux conflits au début de notre git, il a donc fallu comprendre et résoudre ce problème avant de pouvoir disposer d'une base "saine" nous permettant de collaborer confortablement ensemble.

L'étape primordiale après cela était de pouvoir comprendre l'architecture imposée par le dossier fourni, en effet, de nombreuses fonctions et classes étaient déjà implémentés, nous permettant de pouvoir réaliser une multitude d'actions directement.

Nous avons donc débuté par la mise en place des fichiers d'énumérations tels que Orientation et PieceType.

Une fois que nous avions réalisé cette étape avec succès, nous avons décidé de travailler en parallèle sur deux éléments nécessaires pour le projet : Le générateur de grille et l'interface graphique

Generator

Pour la génération d'une grid, premièrement, nous avons implémenté une fonction nous permettant de générer une grid de la taille indiqué par l'utilisateur via la ligne de commande, cette fonction va par la suite choisir aléatoirement une des pièces possibles et leurs assigner une des orientations qu'elle peut prendre. Cette fonctionnalité était surtout nécessaire pour avoir une première approche de la façon de générer une grid car, en effet, cette manière ne permet en aucun cas de générer une grid solvable.

Pour pallier ce problème, nous avons décidé d'implémenter une autre fonction, permettant cette fois ci de générer une grid solvable dès le départ.

Cette fonction récursive commence par la case du coin en haut à gauche (i.e, (0,0)), on va d'abord regarder les pièces qu'il est possible de positionner à cet endroit grâce à la méthode piecesPossible qui va d'abord constituer une liste avec tous les types de pièces et les retirer au fur et à mesure.

Pour les cases en coin, seulement les pièces LTYPE et ONECONN sont possibles car ce sont les seules qui n'auront pas de connecteur qui va pointer hors du tableau (grâce à connectorOutside). Aussi, si la pièce est en bordure de colonne ou de ligne, on supprime le type FOURCONN, car elle aura forcément un connecteur qui pointe dans le vide (hors matrice).

Après cette étape, une sélection va démarrer, on va prendre tous les types de pièces (sauf VOID) et regarder pour chaque orientation leur nombre de mauvais voisins (voisins gênants) qu'on ajoutera dans une liste pour chaque type (le tout dans une Map<PieceType, List<Integer>>). Une fois qu'on aura testé chaque orientation de chaque type dans la case, on va prendre la valeur minimale de chaque type et on va les trier dans un ordre croissant (grâce à TreeMap). Ensuite, on regarde la valeur minimale et tous les types qui ont une valeur strictement supérieure vont être supprimés des possibilités. Il ne reste donc que les pièces possibles.

Une fois qu'on a les pièces possibles, on va shuffle la liste et prendre une au hasard, et on vérifie grâce à isValidOrientation (fonction déjà présente) si l'orientation est valide. Sinon on va tourner la pièce jusqu'à qu'on ait la bonne position.

Ensuite, on va regarder l'emplacement de la prochaine pièce à placer, à l'aide de la méthode getFreeAdjacentCell, qui va regarder les cases voisines vides : si la case est vide et que la pièce actuelle à un connecteur vers cette case, si ce n'est pas le cas, elle est supprimée des cellules adjacentes possibles (représentées par des orientations et converties en coordonnées i,j par orientationToCoords)

De plus, si la pièce est ajoutée à 3 connecteurs ou plus, on va ajouter les coordonnées des emplacements ou la prochaine pièce ne sera pas placée, dans une ArrayDeque. Ces coordonnées seront utilisées pour ajouter des pièces quand la pièce actuelle n'aura plus de cellule adjacente vide mais que la génération n'est pas finie (i.e, toutes les pièces sont fixées sauf celle qui a ses connecteurs vides).

A la fin, la fonction fait un appel récursif avec les coordonnées de la prochaine case choisie. La fonction s'arrête lorsque toutes les pièces sont fixées (tous les connecteurs sont connectés à un voisin)

De cette manière, la grille générée sera solvable, on pourra shuffle l'orientation des pièces et passer la Grid au Solver qui pourra trouver une solution.

Solver & Checker

Par rapport au Solver, nous avons réfléchi, plusieurs possibilités d'orientation sont possibles pour chaque pièce et pour ses voisins. Plusieurs solutions sont éventuellement possibles, l'état but est atteint lorsque toutes les pièces ont leur connecteurs sur celui du voisin. Un des algorithmes utiles pour ce genre de problème de plus court chemin est l'algorithme A*.

En effet, à l'aide de sa fonction heuristique, l'algorithme permet d'aller beaucoup plus vite que les autres algorithmes. Une des heuristiques serait de prendre le nombre de pièces non fixées. Au fur et à mesure qu'on s'approche d'une solution, l'heuristique décroît jusqu'à 0 lorsque l'on est sur l'état but.

Une méthode aussi serait d'utiliser les threads et donner des sous arbres du graphe à chaque thread, ce qui permettrait d'être plus rapide.

Ensuite, la grille serait sauvegardée dans l'emplacement spécifié dans le prompt.

Malheureusement, nous n'avons pas réussi à l'implémenter par manque de temps.

Pour ce qui est du Checker, nous avons d'abord dû implémenter la fonction d'import / export de Grid. Nous avons utilisé les conventions de numéro de type de pièce et d'orientation données dans le sujet. La première ligne d'un fichier est constituée de la hauteur, la seconde de la largeur de la grille. Et sur toutes les autres, on a pour chaque pièce <type,orientation> tous séparés par un espace.

L'utilisateur peut exporter sa grille quand il le souhaite grâce au bouton sur la GUI, et l'importer pour le checker.

Le checker va importer un fichier respectant ces conventions et vérifier si toutes les pièces sont fixées, si c'est le cas, il va afficher SOLVED:true, sinon SOLVED:false.

GUI

Pour la partie de l'interface graphique, nous nous sommes intéressés à comprendre comment il était possible de le réaliser en Java, en effet c'était l'une des premières fois que nous devions en programmer une. Nous nous sommes donc plongés principalement sur les documentations de JFrame, JPanel, gridLayout et JButton qui nous ont permis de coder notre interface aisément, bien que le design est un peu vieillissant.

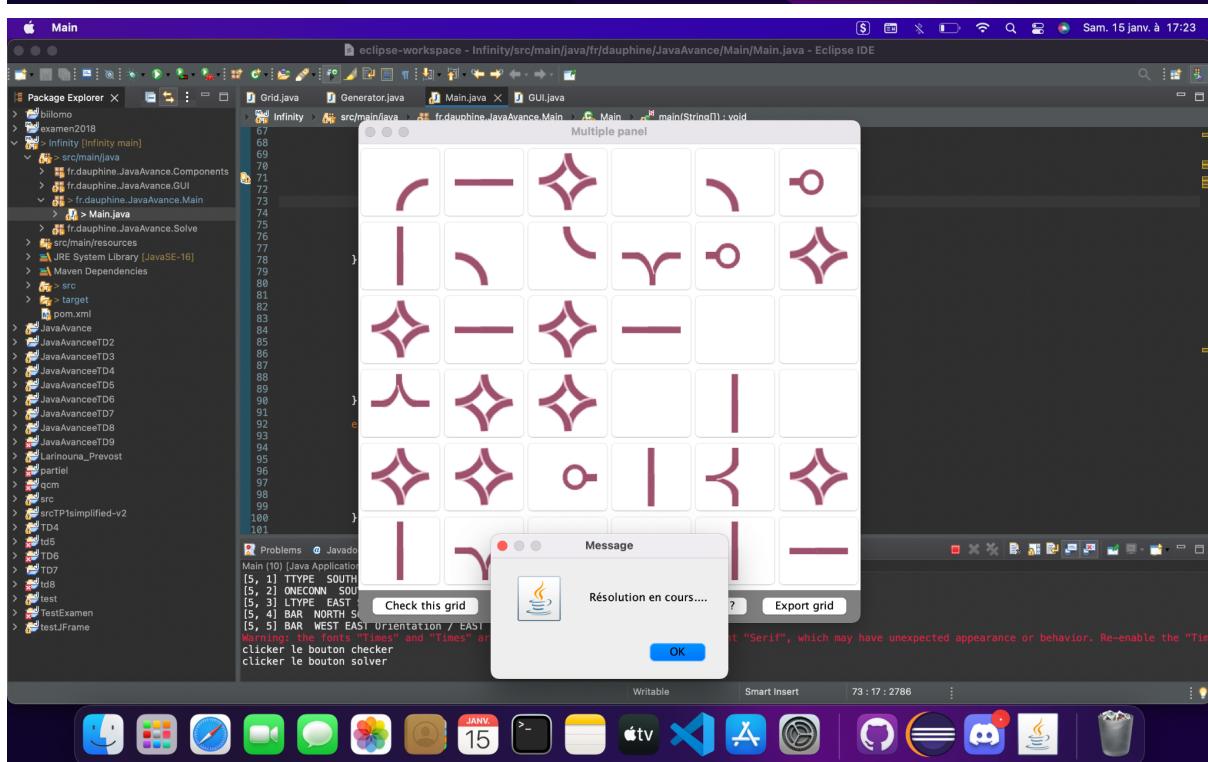
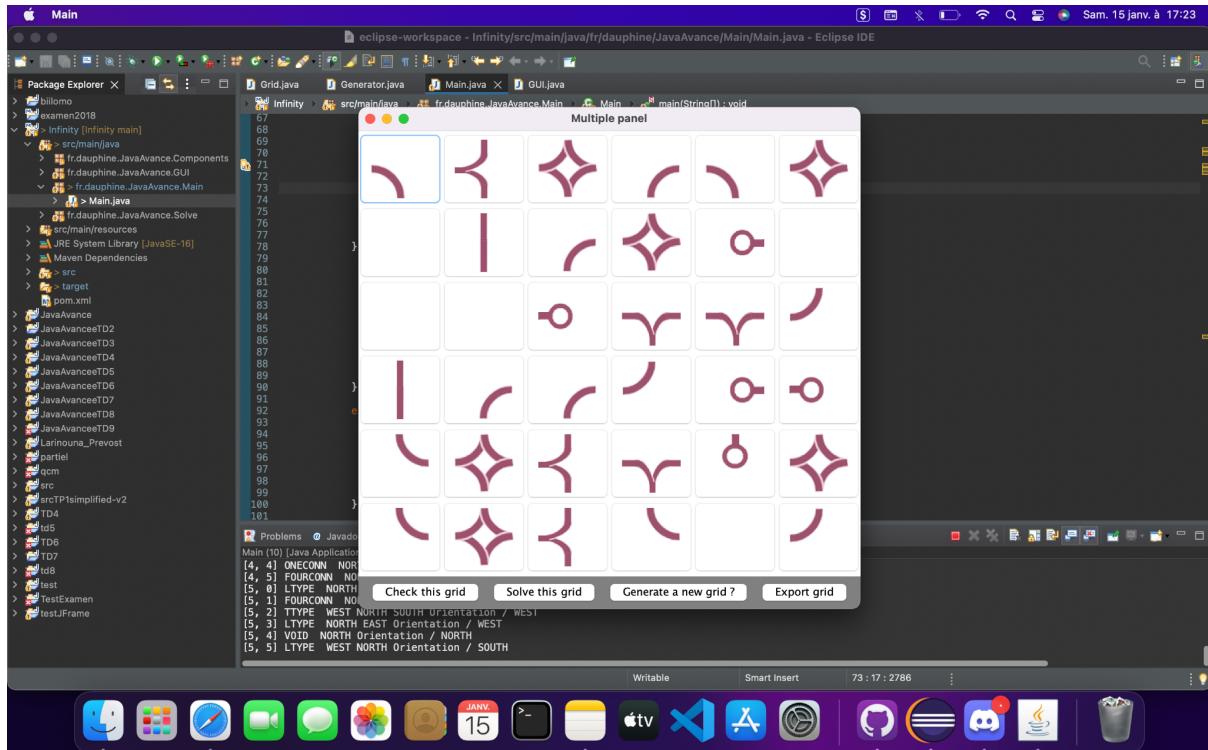
Notre interface graphique se compose donc de deux panels. Le premier contient notre grid avec ces cases et les images de chaque pièce. Le deuxième panel quant à lui permet d'avoir différents boutons comme celui permettant d'afficher une nouvelle grille, celui permettant de lancer le solveur ou le checker et un autre permettant d'exporter notre grid dans un fichier.

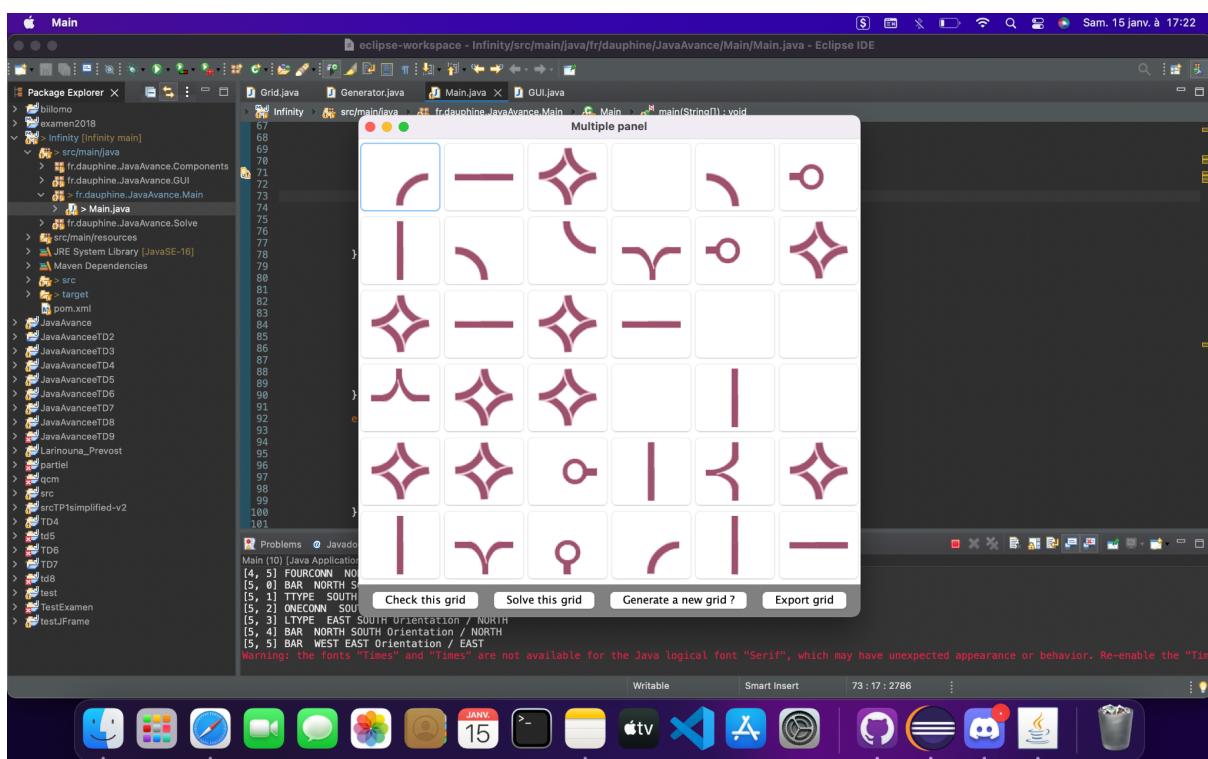
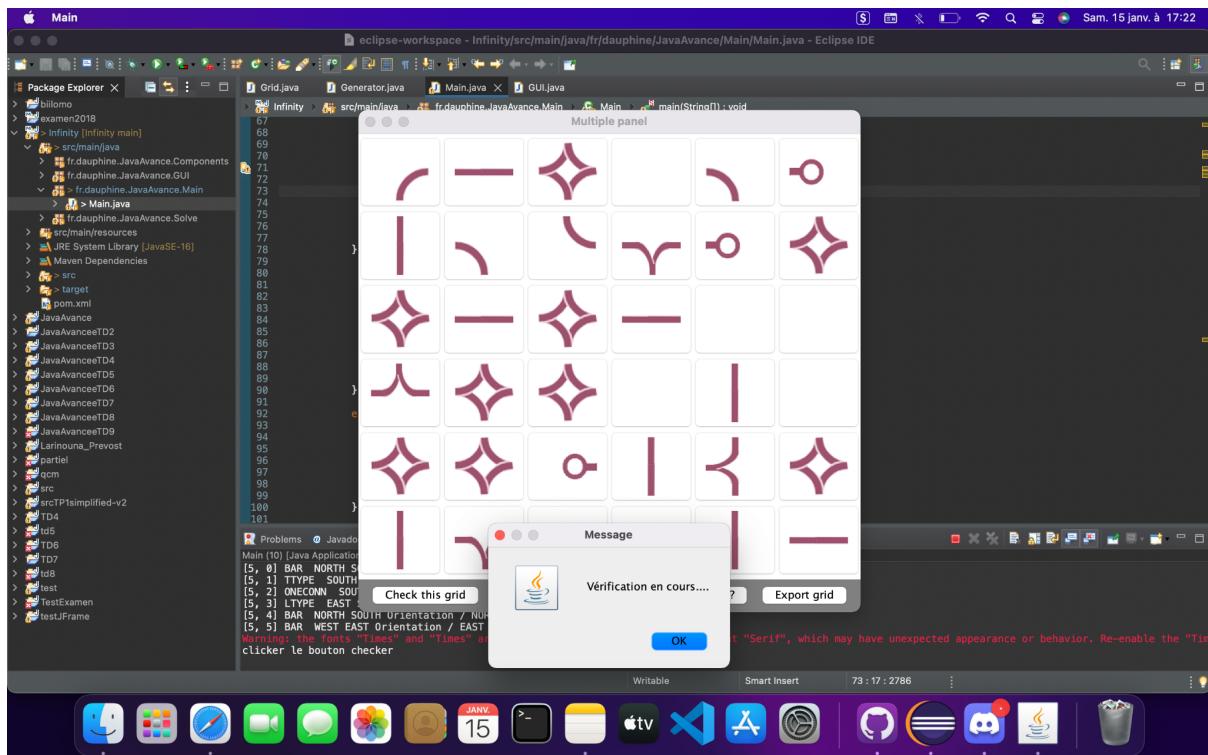
Une autre partie du travail que nous avons réalisé était de pouvoir lancer notre génération de la grille en fonction des paramètres mis au sein de notre ligne commande et donc d'interagir directement grâce à la fonction main.

Pour pouvoir avoir un code lisible et compréhension pour une potentielle évolution dans le futur, nous avons essayé de respecter le plus possible le format de la javaDoc afin de commenter l'ensemble des fonctions que nous avons implémentées, de plus nous avons fait en sorte d'utiliser des noms de fonctions assez claire et compréhensible pour pouvoir les utiliser de manière intuitive. Nous avons aussi pris soin d'essayer de tester les fonctions que nous avons implémentées.

4 - Présentation des résultats et ressenti sur le projet

Afin que ce rapport soit moins monotone, nous avons décidé de présenter quelques vues des frames que nous avons réussi à mettre au point pour le projet de Java.





L'interface nous a permis de pouvoir visualiser plus simplement notre travail et de manière bien plus intuitive.

Les boutons constituent aussi un raccourci pour effectuer une action.

Nous sommes assez fiers de ce que nous avons pu réaliser bien que notre générateur ne soit pas totalement fonctionnel mais le travail fait dessus nous a permis de tester en

condition réelle le développement d'une application java complexe en utilisant une structure déjà mise en place.

Certaines fois, nous avons eu des problèmes avec les fonctions déjà développées, nous nous sommes rendus compte que pour le constructeur de Pièce, l'ordre des paramètres qui devrait être x,y à été inversé pour mettre y d'abord et ensuite x.

Le fait que dans les méthodes <direction>Neighbor, la pièce retourne null, si il y a une pièce de type vide n'était pas correct pour certaines méthode même déjà développées tel que isValidOrientation. Dans cette même méthode, les coordonnées X et Y étaient aussi inversées ce qui a provoqué des bugs.

Ces bugs nous ont fortement ralenti étant donné que nous partions du principe que le code de départ était correct et que nous pensions que c'était principalement nos fonctions qui n'étaient pas correctes.

Nous pensons tout de même que nous aurions pu terminer le projet si nous n'avions pas eu autant de problème. En effet, lors des deux semaines de vacances, un des membres du projet était malade du COVID ne lui permettant pas d'être disponible pour travailler, de plus il a connu certains problèmes familiaux. En parallèle de tout ceci, il était assez compliqué de pouvoir travailler sur le projet et de pouvoir réaliser nos révisions pour les examens terminaux du premier semestre.