Antoine Cornillot

Projet Intégré

Cours de génie logiciel orienté objet

I. Introduction

Dans le cadre du cours de « Génie logiciel orienté objet », nous avons appris à concevoir et à coder une application en UML et en Java. Ce document vise à expliquer la démarche qui nous a permis de réaliser une application fonctionnelle.

Une présentation générale du logiciel sera réalisée dans une première partie intitulée « Cahier des Charges ».

A partir de cette présentation, nous avons identifié les besoins de l'utilisateur vis-à-vis du logiciel, ou comment le logiciel doit se comporter selon les situations.

L'étape suivante a été de conceptualiser l'architecture du logiciel à partir de ces contraintes extérieures.

Finalement nous avons développé le code source du projet. Nous reviendrons sur l'organisation temporelle de l'organisation ainsi que sur les tests ayant permis de définir l'application comme fonctionnelle.

Nous terminerons par un bilan du projet et une ouverture sur les perspectives envisageables pour le logiciel.

II. Cahier des charges

L'objectif de ce projet est de coder en Java une application similaire au jeu « Flow Colors ».

Dans ce jeu, le joueur est face à une grille dont certaines cases sont colorées.

Son but est de relier deux cases de même couleur sans que les chemins de couleurs différentes ne se coupent. Le jeu est considéré comme fini lorsque chaque case de la grille appartient à un chemin.

Sur le modèle en ligne, le tracé des chemins se fait à la pointe de la souris. Dans notre logiciel, l'interface est légèrement différente : le joueur sélectionne un point coloré sur la grille et déplace l'extrémité du chemin associé avec les flèches du clavier.

Un clic sur une case vide ou utilisée par une ligne n'a donc aucun effet. On décide qu'il n'est pas possible de faire progresser une ligne sur une case déjà occupée par une autre ligne.

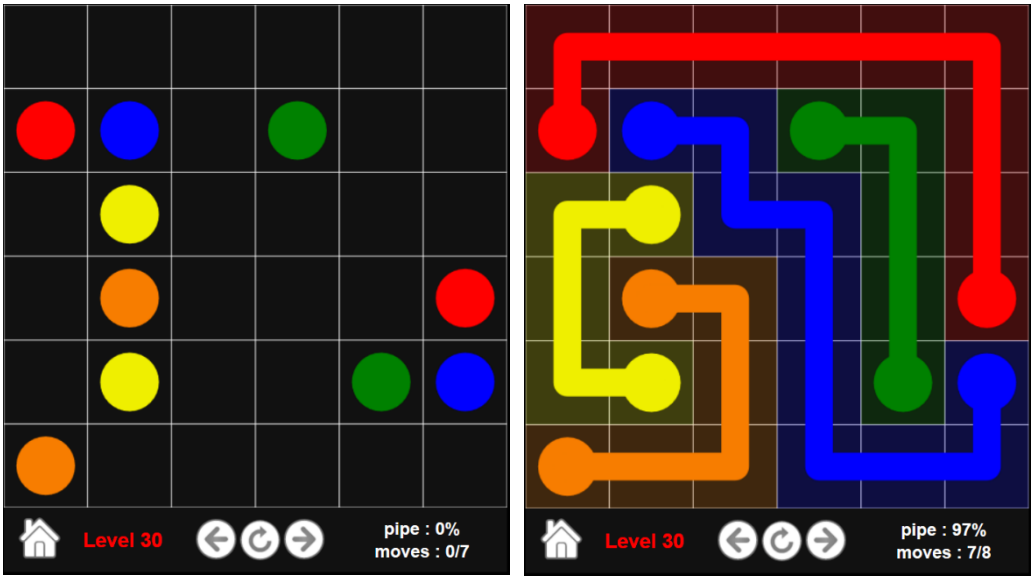

Figure 1 : Exemple de situation de départ (gauche) et exemple de résolution du jeu (droite)

III. Besoins de l'utilisateur

Lorsque l'utilisateur est face à la grille de départ, il souhaite pouvoir sélectionner une extrémité et modifier un chemin avec les flèches de son clavier. L'utilisateur souhaite aussi savoir si la solution qu'il propose résout le jeu.

Finalement le joueur peut être à la recherche d'une solution optimale : pour cela, il demandera le nombre de mouvements qu'il a pu réaliser depuis le début de la partie.

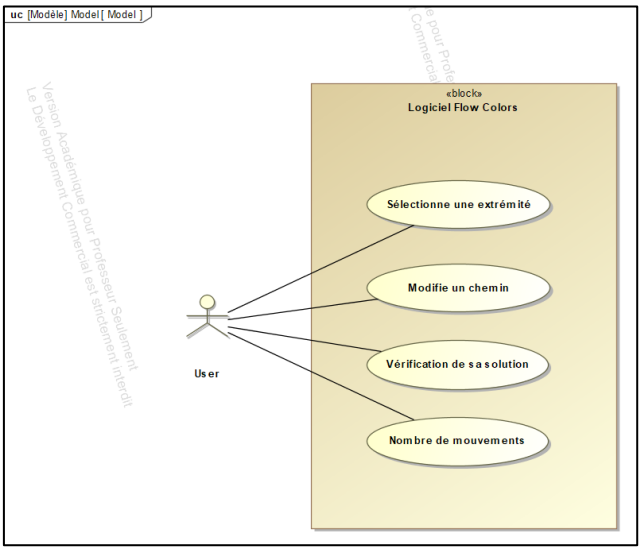

Figure 2 : Diagramme d'utilisation de l'application « Flow Colors »

Du point de vue du joueur, le jeu se comporte ainsi :

- La fenêtre s'ouvre lorsque le joueur décide de jouer.
- Le jeu est alors en attente de commande de la part du joueur. Toute commande avec les flèches sera sans conséquence puisqu'il n'y a pas d'objet courant.
- Si le joueur sélectionne une extrémité, le jeu prend comme objet courant le chemin sélectionné. Ceci n'entraîne pas de modification du chemin tant que le joueur ne clique pas sur une touche du clavier.
- Lorsque le joueur clique sur une case qui n'est pas une extrémité, le jeu redevient « idle ».
- Le jeu se termine lorsque la progression du joueur est de 100 %.

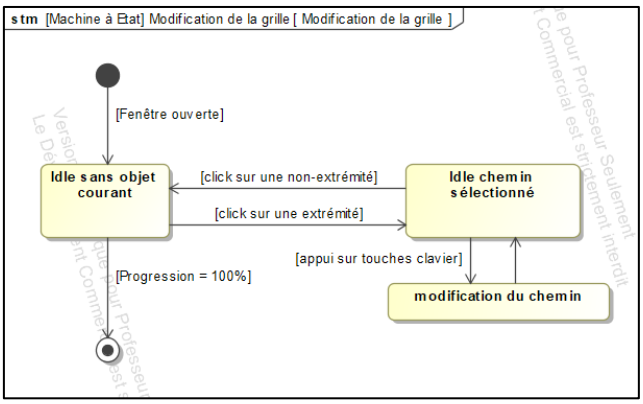

Figure 3 : Diagramme d'état de l'interface selon les commandes de l'utilisateur

IV. Conception

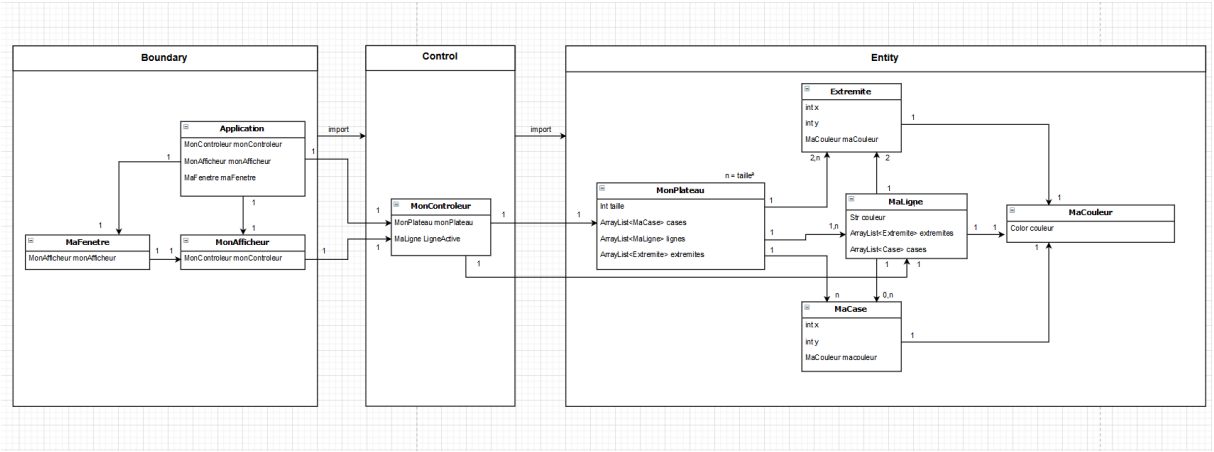

Figure 4 : Diagramme de classe du logiciel

Le code se structure en trois packages. Le package « Boundary » réalise l'interface entre l'utilisateur et la machine. Il n'a accès qu'au package « Control », dont la classe « MonControleur » représente l'état de notre plateau au fil des commandes de l'utilisateur. Ce package a une connaissance des éléments constituant le plateau, dont les classes sont regroupées dans le package « Entity ».

IV.1) Le package « Entity »

A partir des besoins de l'utilisateur, on distingue cinq classes différentes. Nous souhaitons réaliser un plateau constitué d'un nombre de cases déterminé par sa taille. Ce plateau contiendra au moins deux extrémités qui seront des cases dont la couleur ne pourra être modifiée. Sur ce plateau nous tracerons des lignes entre les extrémités.

- Une couleur est définie par une couleur au sens du module « Color » de Java. Lorsque l'on manipule la classe MaCouleur, on manipule donc des objets « Color ».
- Une case est caractérisée par sa position sur l'axe des abscisses x et sa position sur l'axe des ordonnées y. On souhaite pouvoir accéder à ses coordonnées, mais elles doivent restées inchangées. De base, elles ont toutes la même couleur (noire), mais on souhaite pouvoir la modifier.
- Une extrémité s'apparente à une case, mais c'est en fait plus complexe. Une extrémité est elle aussi caractérisée par sa position sur le plateau, mais elle possède en plus une couleur propre non-modifiable.
- Une ligne est caractérisée par deux extrémités que l'on souhaite joindre, ainsi que par une couleur. Entre les deux extrémités, on trace une pile de cases côte-à-côte.
- Le plateau est défini par des caractères invariants dans le temps : sa taille et ses extrémités. On peut alors générer un plateau de départ en créant des lignes initialement définies à partir de deux extrémités de même couleur, de cette couleur et d'une liste de cases vides.

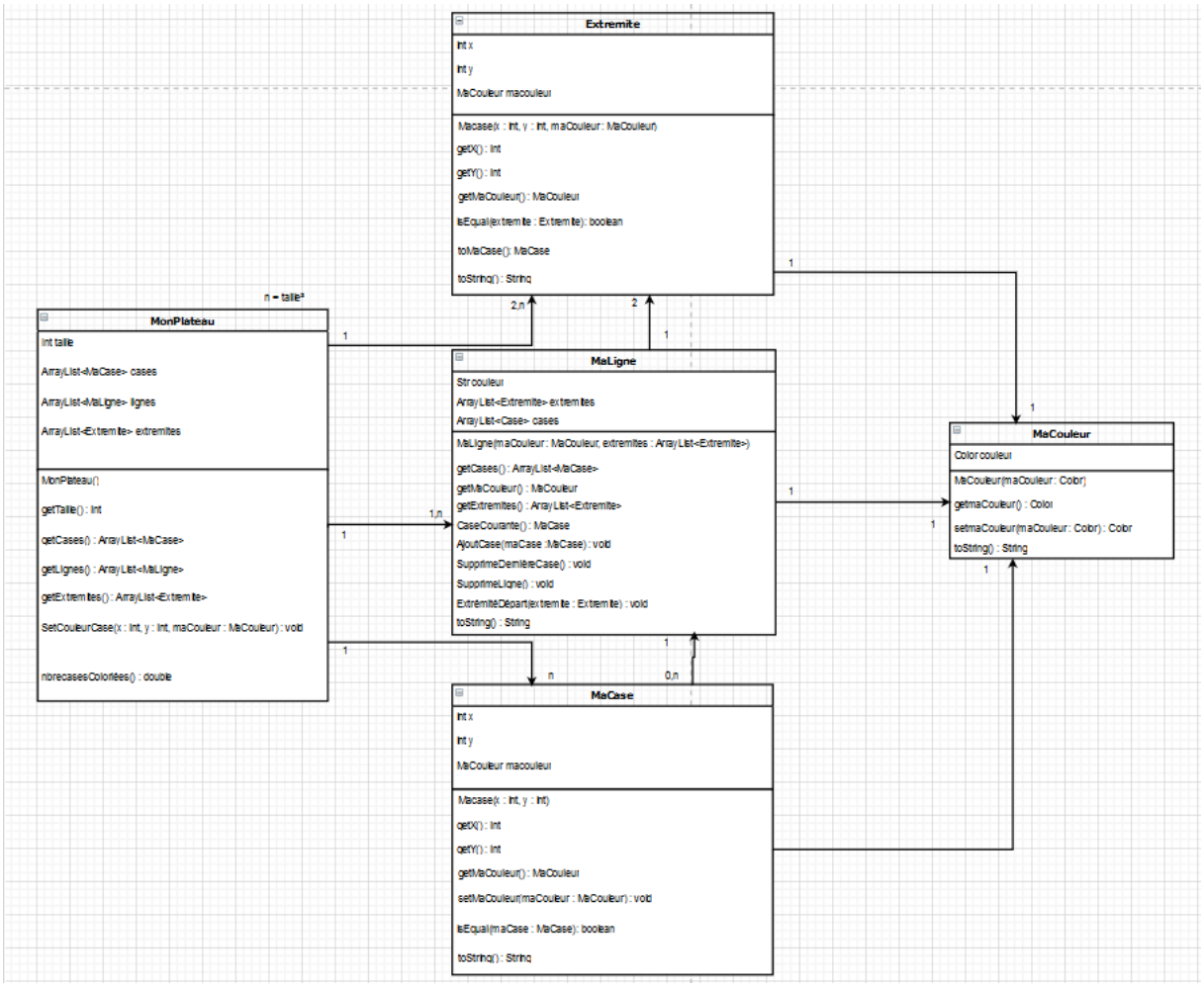

Figure 5 : Diagramme de classe du package « Entity »

IV.2) Le package Boundary et le contrôleur

Finalement le logiciel contient une classe avec une méthode main : la classe « Application » crée un contrôleur, un afficheur et une fenêtre.

- La classe « MonControleur » (le contrôleur) contient toutes les informations auxquelles l'application souhaite accéder pour fonctionner. Le contrôleur est ainsi initialisé avec un plateau dont la taille et la position des extrémités ne va jamais changer. Par contre, l'utilisateur va modifier la couleur des cases de ce tableau ainsi que ses lignes. Pour savoir quelle est la ligne modifiée par l'utilisateur, on crée un objet « MaLigne » que l'on appelle « LigneActive ». On l'initialise comme un objet null. On veut régulièrement savoir quelle est cette ligne ainsi que la modifier.
- La classe « MonAfficheur prend en entrée un objet « MonControleur ». C'est une sous-classe de la classe JPanel. On configure sa méthode paint pour qu'elle construise le plateau graphiquement. Elle construit des cases de taille 100x100 pixels, et rajoute à droite du plateau des informations concernant la progression du joueur, ainsi que la couleur de la ligne active (cette couleur est noire par défaut). Après chaque modification du plateau, on construira graphiquement le nouveau plateau avec la méthode repaint(). Les méthodes up(), down(), right(), left() modifient le contrôleur dans le cas où l'utilisateur demande à ce que la case courante soit la case du dessus, du dessous, de droite ou de gauche. La

méthode getPointer(x : int, y : int) modifie le contrôleur en fonction de l'endroit où l'utilisateur a cliqué (joue principalement sur la ligne active).

- La classe MaFenetre est une sous-classe de JFrame. Elle génère une fenêtre d'une certaine taille avec le nom de l'application « ColorFlow ». Sur cette fenêtre, elle rajoute les dessins de l'afficheur qu'elle prend en entrée (la construction graphique du plateau). Cette classe contient aussi les instances des entrées clavier et souris. On crée des méthodes qui lancent les opérations de l'afficheur en fonction de ces entrées.

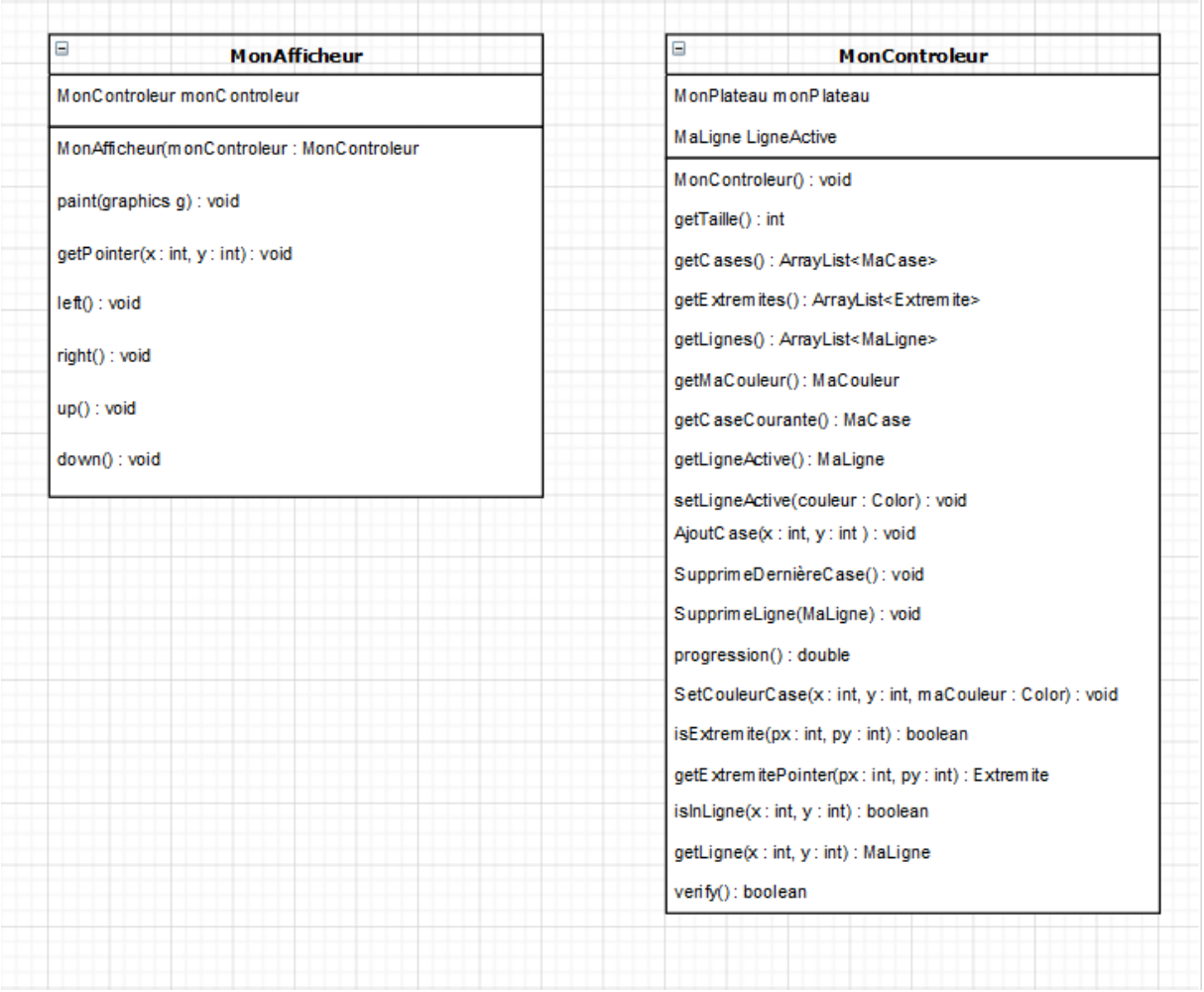

Figure 6 : Diagramme de classe du contrôleur et de l'afficheur

V) Réalisation :

J'ai réalisé ce logiciel en suivant une certaine organisation.

J'ai tout d'abord défini le cahier des charges et les besoins de l'utilisateur. J'ai ensuite réalisé un premier modèle métier du logiciel, afin de définir les classes dont j'avais besoin, leur constructeur et leur getter et setter.

J'ai commencé par réaliser les classes, en commençant par le package entity, puis le package control et finalement le package boundary. J'ai réalisé une classe de test pour chaque classe créée afin de voir que vérifier mon code au fur et à mesure que je l'avançais. J'ai réalisé des tests de réalisation de la grille en utilisant le package testswing.

Je suis ensuite parti de ce que souhaite l'utilisateur pour définir les méthodes de l'afficheur, puis du contrôleur. J'ai donc dû rajouter des méthodes aux classes du package entity.

Pour finir j'ai réalisé des tests sur le jeu, et j'ai dû rajouter des conditions dans mes méthodes afin d'éviter quelques erreurs (gestion d'attributs null, léger décallage dans les pixels à cause de la barre d'affichage, …). Je me suis aussi rendu compte de la nécessité de rajouter des informations sur la fenêtre concernant la couleur active et le taux de progression. Pour améliorer le rendu utilisateur, j'ai aussi rajouté un affichage final lorsque la grille est complète (j'ai donc rajouté la méthode verify()).

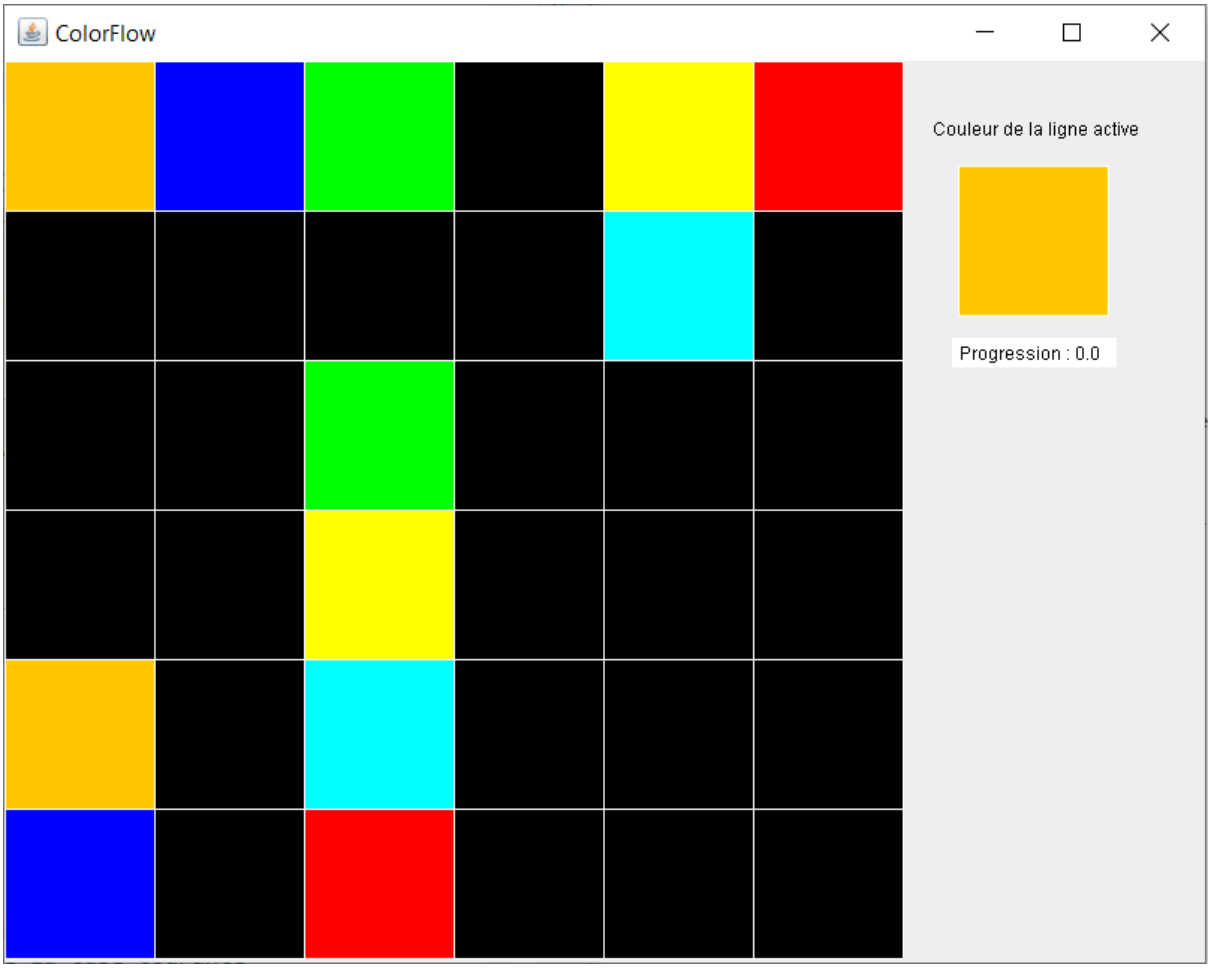

Figure 7 : Situation initiale du jeu

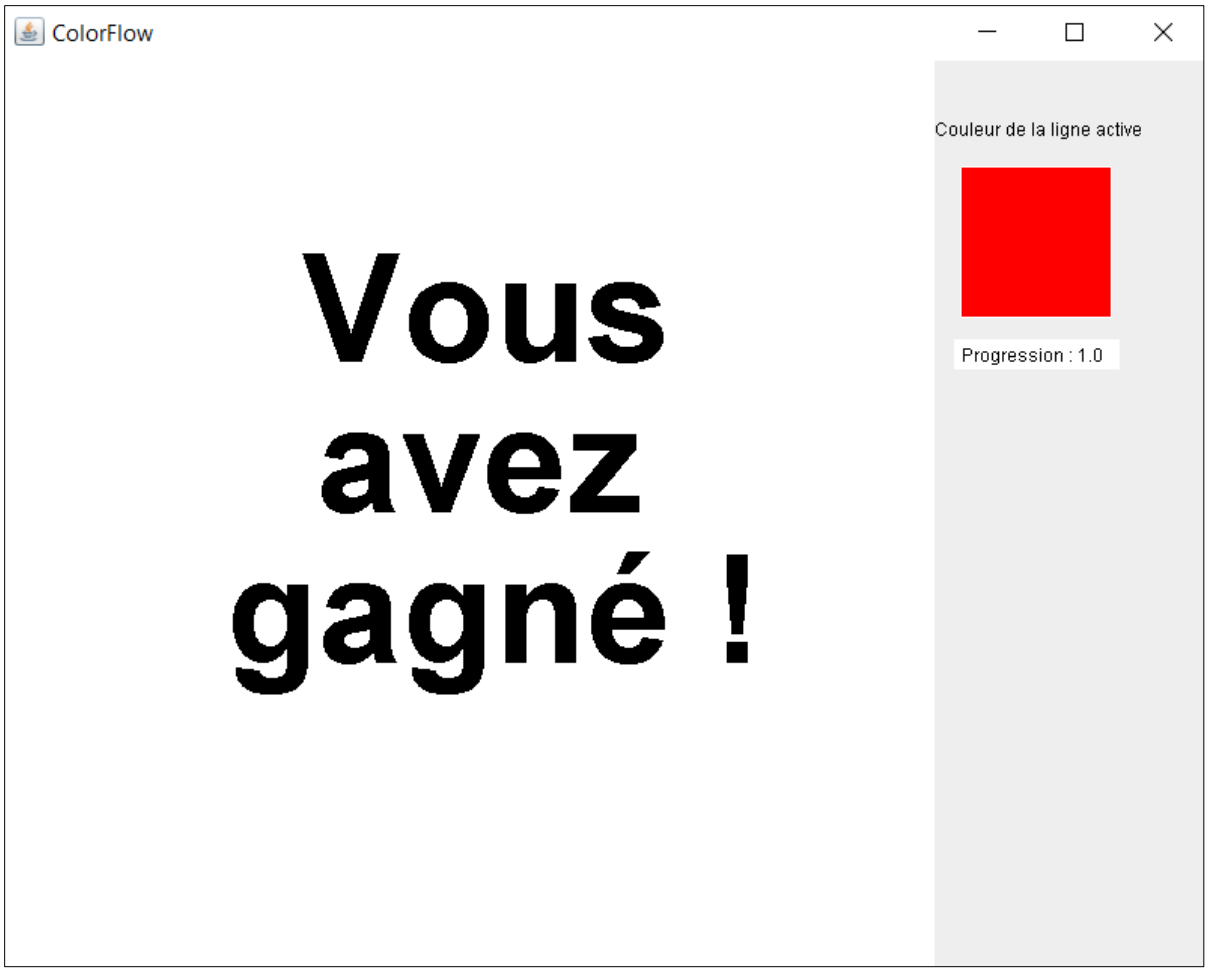

Figure 8 : Affichage final lorsque jeu est fini

VI) Guide d'utilisation :

L'application ne se compose que d'un seul mode de jeu. On lance l'application en ouvrant le projet dans Eclipse, en se positionnant dans la classe « MonApplication » et effectuant « Run As… » « 1 Java Application » dans le menu déroulant de Eclipse.

Initialement, aucune ligne n'est sélectionnée. Le joueur doit sélectionner la case de couleur à partir de laquelle il souhaite tracer un chemin. En haut à droite, la case « couleur de la ligne active » prend la couleur de cette case. Le jeu est inactif à chaque fois que cette case est noire.

Le joueur trace ensuite son chemin son plateau en utilisant les flèches directionnelles. Il doit joindre deux cases de même couleur sans couper un autre chemin. Les cases initialement colorées sur le plateau ne peuvent changer de couleur. Si le chemin en cours de tracé coupe un autre chemin, ce dernier est effacé.

Si, au cours de la partie, le joueur sélectionne une case à partir à partir de laquelle un chemin est déjà tracé, il reprend le tracé à partir de l'autre extrémité du chemin. S'il sélectionne l'autre extrémité, le chemin est effacé et il commence à tracer un chemin depuis cette nouvelle case.

Le jeu est terminé lorsque chaque chemin joint deux extrémités et que toutes les cases sont coloriées (soit que la progression est de 1.0).

VII) Conclusion

Le logiciel réalisé remplit le cahier des charges. Les règles sont appliquées, et le jeu semble afficher que le joueur a gagné seulement lorsqu'il a réellement gagné.

Des améliorations restent envisageables. Le joueur pourrait commencer à jouer à partir d'un menu principal. Il pourrait avoir la possibilité de choisir parmi plusieurs grilles de départ, comme sur le jeu en ligne. L'expérience utilisateur pourrait être améliorée en différenciant les cases extrémités des autres cases. Certains utilisateurs pourraient être intéressés par savoir le nombre de coups qu'ils ont réalisé ou le temps qu'ils ont mis à remplir la grille.

Le code aurait pu être bien différent. Un attribut direction aurait pu être ajouter à la classe case afin de pouvoir tracer des lignes fines – comme sur l'exemple en ligne, à la place de colorier la case en entier. Plutôt que d'utiliser l'interface graphique Swing, l'état du contrôleur aurait pu être affiché avec des « println() » et les actions lues avec un « Scanner ». Une piste majeure d'amélioration pourrait être d'implémenter un code permettant de colorier les cases en glissant dessus avec la souris.