Génie logiciel orienté objet

Projet intégré – Flow free

I. Introduction

Dans le cadre du projet intégré en génie logiciel orienté objet, nous avons réalisé en binôme le jeu *Flow free*. Ce document à pour but de présenter le travail effectué, la démarche utilisé lors du développement et le résultat final obtenu.

Nous allons donc définir le cahier des charges qui devra être satisfait, nous allons préciser le comportement attendu du programme en réponse aux instructions d'un joueur, ensuite nous présenterons le fonctionnement interne du programme lors de son exécution ainsi que les liens entre les différentes classes qui le composent. Nous allons détailler notre méthode de travail et nous finirons sur les perspectives d'amélioration du logiciel.

II. Cahier des charges

Le jeu *Flow free* consiste à relier des paires de plots de couleur identique placés sur une grille sans passer deux fois sur la même case et en remplissant toute la grille.

Le jeu doit donc se présenter sous la forme d'une grille (le plateau) composée de cases dont certaines sont occupées par un plot, avec exactement deux plots de chaque couleur.

Le jeu se termine lorsque le plateau est complet, i.e chaque case est occupée par un plot ou un tronçon de tuyau qui relie deux plots de même couleur, et toutes les paires de plots sont reliées.

Lorsque le joueur clique sur un plot, le jeu crée le tuyau de la couleur du plot au départ de celui-ci et le joueur peut l'allonger à l'aide des flèches de direction.

Lorsque qu'un tuyau est en cours et que le joueur presse une direction, un tronçon est ajouté à l'extrémité du tuyau en cours dans la direction sélectionnée, à condition que la case désignée soit vide.

Cas particulier si la direction choisie pointe à contresens du tuyau en cours, i.e vers le tronçon précédent, alors le dernier tronçon du tuyau est supprimé.

Il est impossible créer un tronçon sur une case occupée par un autre tronçon, un plot d'une couleur différente, ou située hors du plateau. Si la case est occupée par le second plot de la couleur du tuyau, celui-ci est alors terminé.

Une représentation graphique permet au joueur une lecture immédiate de l'état actuel du plateau de jeu et lui permet également d'interagir avec le jeu (détection de la souris, clics, touches du clavier pressées).

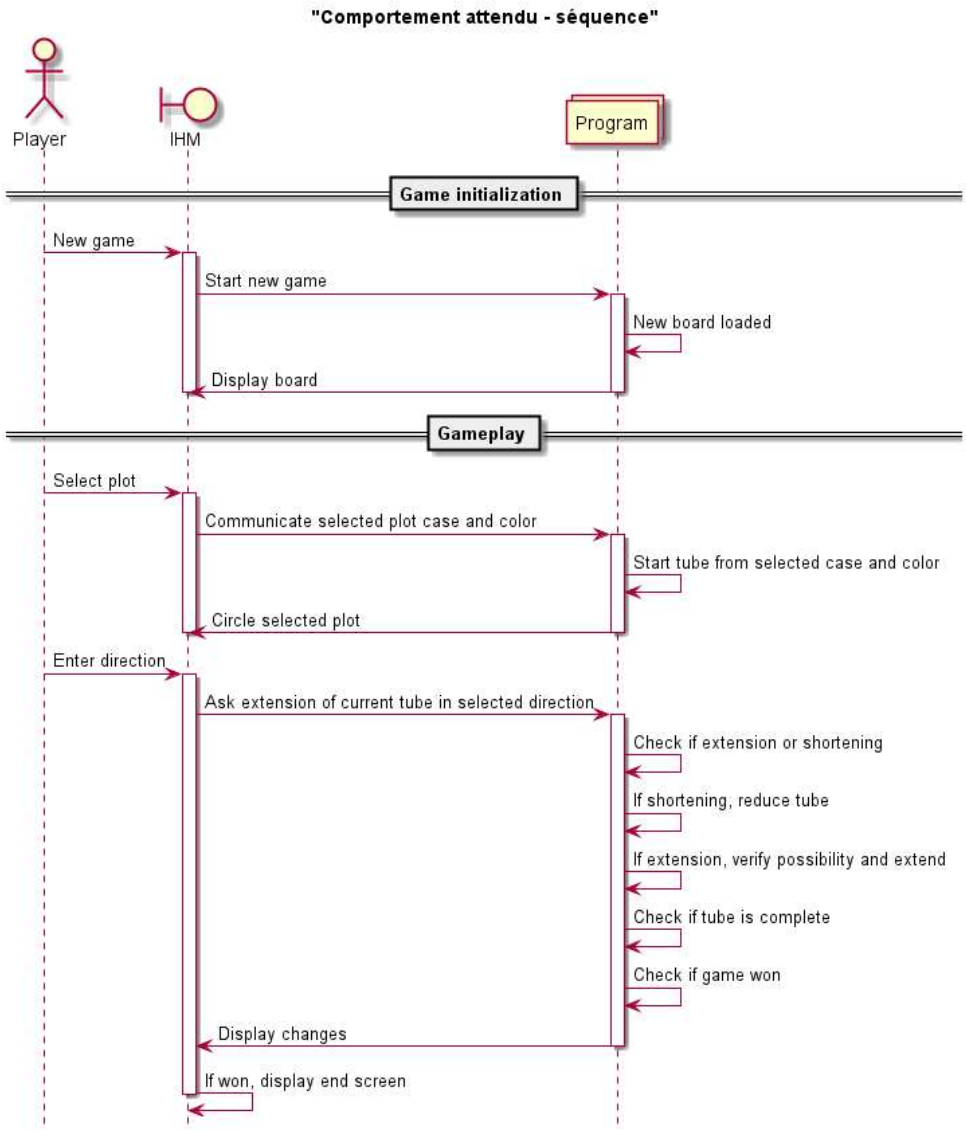
Un écran de fin de partie s'affiche lorsque le niveau est terminé.

Le jeu comporte une banque de différents niveaux, à chaque nouvelle partie l'un d'eux est choisi aléatoirement.

III. Expression des besoins

Au cours d'une partie, le joueur envoie des instructions au jeu et celui-ci modifie le plateau en fonction des ces instructions et en suivant le cahier des charges, il affiche le nouvel état du jeu dans l'interface graphique pour que le joueur en prenne connaissance.

Une séquence de jeu du point de vue du joueur peut être représentée par le diagramme suivant :

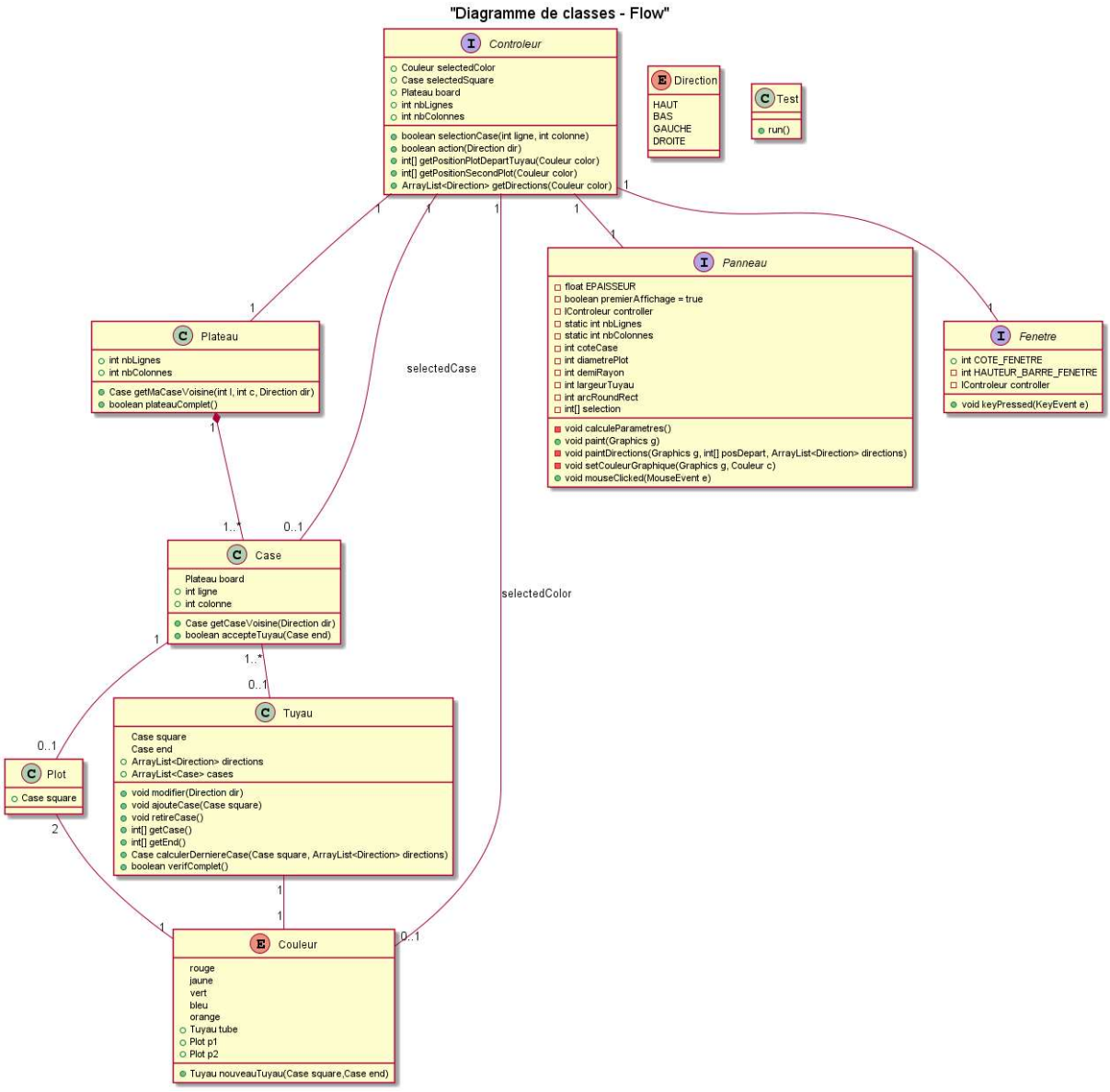


Dans le cas où le joueur saisi une commande non reconnue par le jeu, ou si il donne une instruction irréalisable (par exemple prolonger un tuyau sur une case déjà occupée par un autre tuyau), le jeu reste dans le même état et le joueur doit saisir une nouvelle commande pour continuer à jouer.

IV. Conception

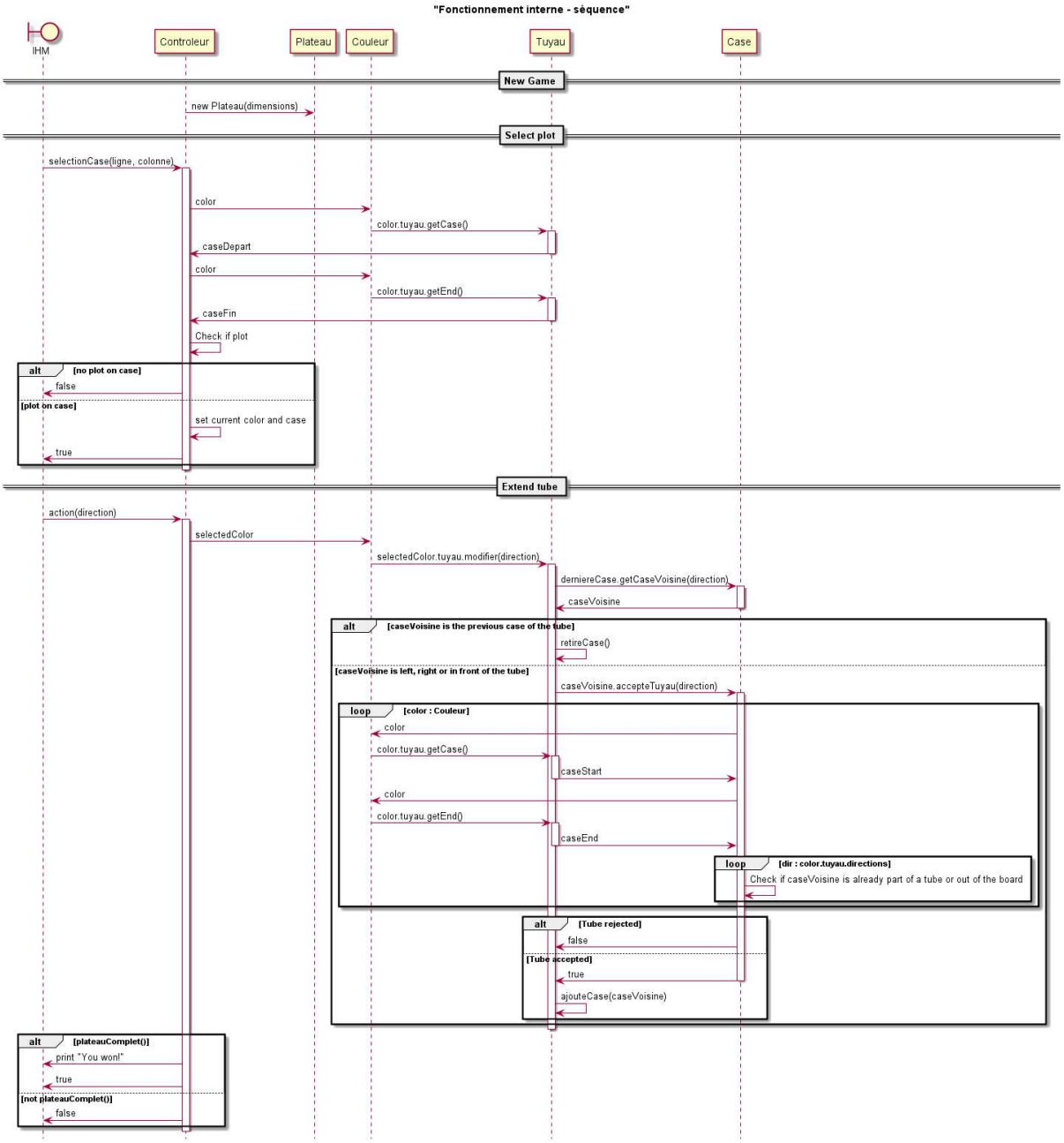
Le programme est organisé en classes qui représente les différentes entités qui composent le jeu et permettent de définir leurs attributs et méthodes respectifs. Les instances de ces classes sont amenées à échanger des informations et des instructions.

L'architecture que nous avons choisie peut être représentée par le diagramme de classes suivant :



Les instructions du joueur sont modélisées par une classe *Controleur* qui communique avec la classe *Plateau* pour sélectionner une case suite à une action du joueur. La classe *Plateau* cherche si un plot se trouve sur la case et définie la couleur du tuyau à créer le cas échéant. La classe *Controleur* communique aussi avec la classe *Couleur* afin d'accéder au tuyau d'une couleur et demande à l'instance de la classe *Tuyau* de se modifier en fonction de l'instruction du joueur. La classe *Tuyau* communique avec la classe *Case* pour déterminer la possibilité d'occupation de la case suivante. Les classes *Fenetre* et *Panneau* servent d'interface graphique et font le lien entre la saisie du joueur et la classe *Controleur*.

Le diagramme de séquence suivant représente le fonctionnement du programme :



V. Réalisation

Pour mener à bien ce projet, nous avons dans un premier temps réfléchi séparément sur notre vision du jeu et de l'organisation du code en classes. Nous avons ensuite mis en commun nos réflexions et nous avons choisi l'architecture que nous allions adopter.

Nous avons mis en place un Git pour faciliter le travail collaboratif, permettre de versionner notre code et avoir accès à la dernière version du projet à tout moment lorsque nous travaillions séparément.

Nous avons réalisé un premier diagramme de classe en s'inspirant de l'exemple donné et en l'enrichissant des méthodes que nous avions déjà identifiées. Nous avons commencé le développement des différentes classes en suivant la description fournie par le diagramme. A un certain stade du développement, nous avons été amenés à revenir sur certains partis pris dans le diagramme de classe afin de simplifier certaines méthodes. Nous avons donc modifié l'architecture du code et le diagramme de classes en conséquence. Une fois le jeu fonctionnel, nous avons déroulé une partie en allant lire dans le code les différentes méthodes appelées successivement par la saisie d'une instruction afin de construire le diagramme de séquence du fonctionnement interne du programme, en vérifiant bien que les différents cas de figures sont traités. Nous avons finalement ajouté différents niveaux jouables.

VI. Conclusion

Au terme de ce projet, nous avons développé de jeu *Flow free* en langage Java, avec une description de l'architecture du programme et de son fonctionnement grâce à des diagrammes UML. Le jeu comporte une interface graphique qui permet d'y jouer de manière simple et intuitive à n'importe lequel des niveaux implémentés.

Dans une perspective d'amélioration du programme, nous pourrions faire en sorte que le programme propose automatiquement de rejouer lorsqu'une partie est terminée, nous pourrions créer une application exécutable pour lancer le jeu, développer une interface permettant au joueur de choisir son niveau, proposer un classement par difficulté ou encore mémoriser le meilleur score du joueur (temps, nombre de coup…).

Certains diagrammes présentés dans ce rapport sont denses en information et peuvent être difficiles à lire à l'échelle de ce document, vous pouvez également les retrouver au format png dans le dossier UML du projet.