Rapport de projet

Jeu FlowFree

Ce document présente les différentes phases du projet Flow Free réalisé dans le cadre de l'électif Génie logiciel orienté objet

Dossier rédigé par *Hugo BORSONI et Vincent FLATTOT.*

17/01/2021

TABLE DES MATIERES

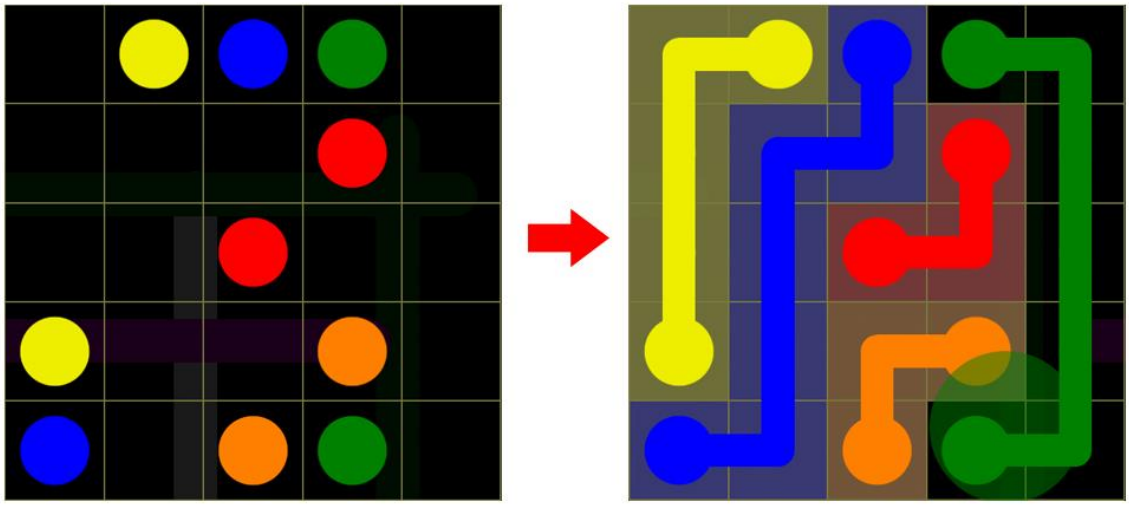
NB : les codes utilisés pour dessiner les différents diagrammes sont fournis dans le dossier code_dossier rendu avec ce fichier. On pourra par exemple les entrer dans le site web plantuml.com afin d'obtenir les résultats graphiques.

01. Cahier des charges

A. Presentation du jeu Flow Free®

Le jeu dont nous nous inspirons est Flow Free®. Il s'agit d'un jeu de puzzle développé par le studio Américain Big Duck Games et sortit en juin 2012. Il s'agit initialement d'un jeu pour mobile et est disponible sur IOS et Android.

Le concept du jeu est assez simple. On part d'un plateau carré découpé en NxN cases. On dispose sur 2xN cases des « Plots » en forme de ronds. Il faut ensuite à l'aide de « Tuyaux colorés » réussir à relier chaque plot de la même couleur tout en s'assurant que l'ensemble {Tuyau+Plot} couvre bien l'ensemble des cases du Plateau et qu'aucun Tuyau ne croise un autre Tuyau.



Une illustration d'une partie au début et à a fin

B. Presentation d'une phase de jeu (version mobile)

Dans la phase de jeu, l'utilisateur peut cliquer sur un plot puis tracer avec son doigt le chemin qu'il souhaite lui faire prendre. **Dans notre version, l'utilisateur cliquera avec sa souris sur un plot puis déplacera le tuyau grâce aux flèches de son clavier**

Si l'utilisateur lâche son doigt, le tuyau s'arrête sur la dernière case balayée par l'utilisateur. **De même, dans notre version, nous utiliserons ce procédé**

Si l'utilisateur fait croiser le nouveau tuyau avec un ancien, l'ancien tuyau est effacé jusqu'à la case précédant le croisement, le tuyau en construction reste celui que l'utilisateur était en

train de construire. **Dans notre version du jeu, nous supprimerons le tuyau courant entièrement et l'utilisateur pourra recliquer sur un plot pour construire un nouveau tuyau**.

Durant tout le jeu, l'utilisateur est informé du nombre de tuyaux qu'il a réussi à finir, le pourcentage du plateau couvert par les tuyaux et le nombre de coup réalisé. **Nous avons reproduit ce comportement.**

Lorsque l'utilisateur à réalisé le dernier mouvement avant la victoire, le jeu l'informe de la réussite et indique le nombre de tentatives qu'il a fallu pour réussir le niveau**. Nous avons reproduit ce comportement, en considérant comme une action chaque frappe sur le clavier qui donne lieu à la modification d'un Tuyau.**

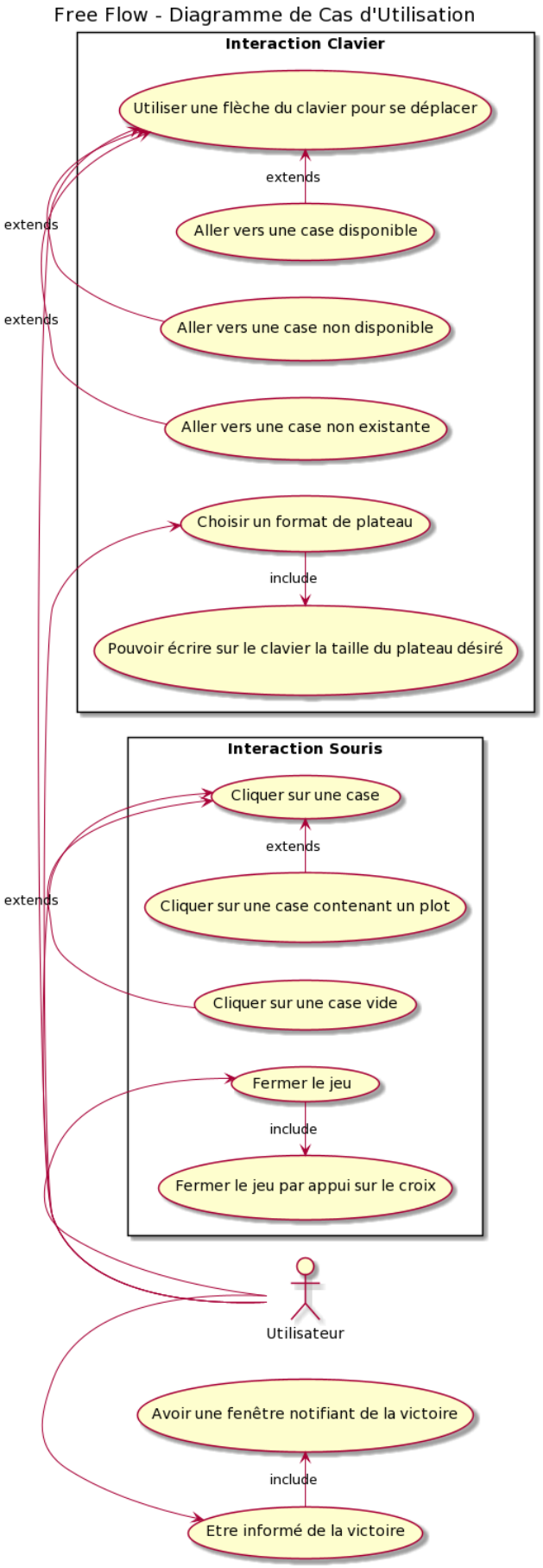
C. Notion de niveaux

Dans le jeu, il existe différents niveaux auxquels on peut accéder lorsqu'on a réussi les niveaux précédents. **On donne dans ce projet la possibilité à l'utilisateur de changer de « niveau » en changeant le nombre de cases du plateau**. **Ce choix s'effectuera au début via un sélecteur.**

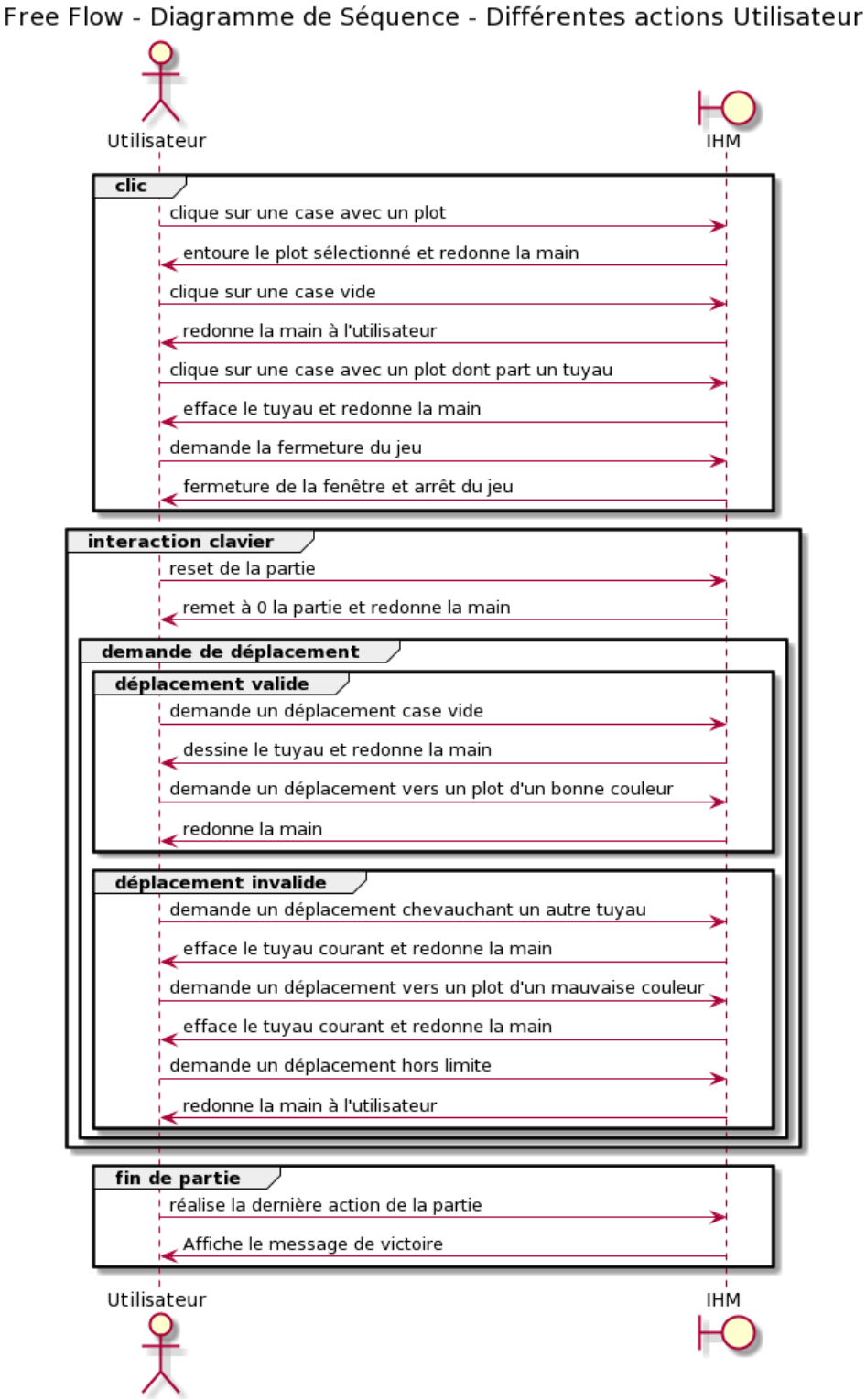
02. Expression des besoins

L'utilisateur du jeu a plusieurs attentes que nous récapitulons ici sous forme d'une liste et que nous illustrons via un diagramme de cas d'utilisation et de différents scénarios.

- Pouvoir sélectionner un plot pour créer un tuyau
- Pouvoir se déplacer dans le plateau avec les tuyaux
- Pouvoir connaitre son avancée dans le jeu
- Pouvoir savoir qu'on a gagné
- Pouvoir quitter le jeu

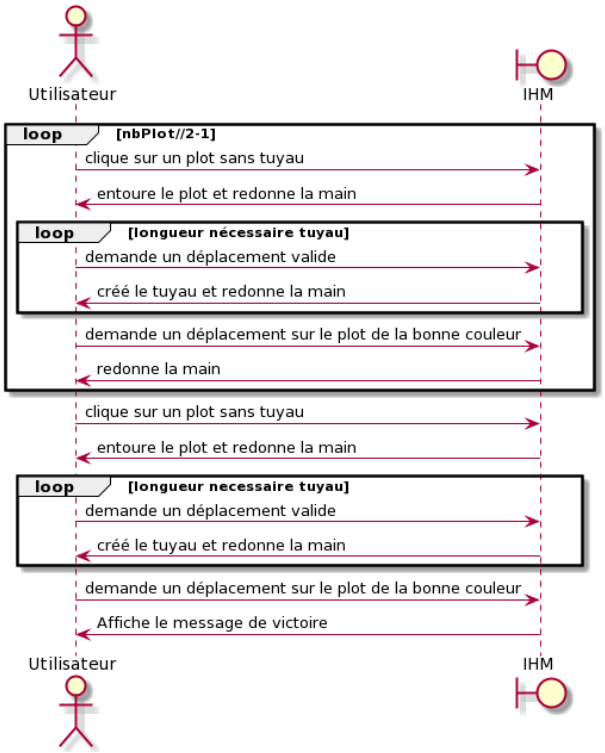

Free Flow - Diagramme de Cas d'Utilisation
Interaction Clavier
Utiliser une flèche du clavier pour se déplacer
extends
Aller vers une case disponible
extends
Aller vers une case non disponible
extends
Aller vers une case non existante
Choisir un format de plateau
include
Pouvoir écrire sur le clavier la taille du plateau désiré
Interaction Souris
Cliquer sur une case
extends
Cliquer sur une case contenant un plot
extends
Cliquer sur une case vide
Fermer le jeu
include
Fermer le jeu par appui sur le croix
Utilisateur
Avoir une fenêtre notifiant de la victoire
include
Etre informé de la victoire

Ces cas d'utilisations peuvent être mis en parallèle avec les actions qu'ils vont nécessiter de la part de l'Interface Homme-Machine :

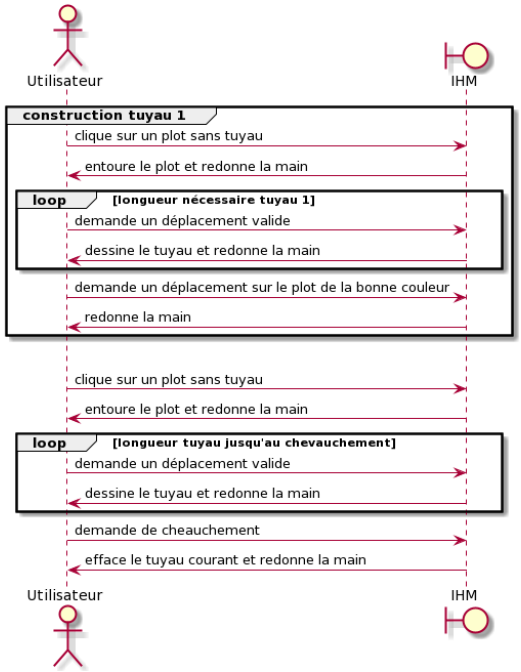


Afin de permettre une meilleure compréhension de ces éléments, on propose la modélisation d'un scénario « parfait » et d'un scénario avec une erreur du point de vue du joueur

Free Flow - Diagramme de Séquence - Scénario parfait(vue utilisateur)



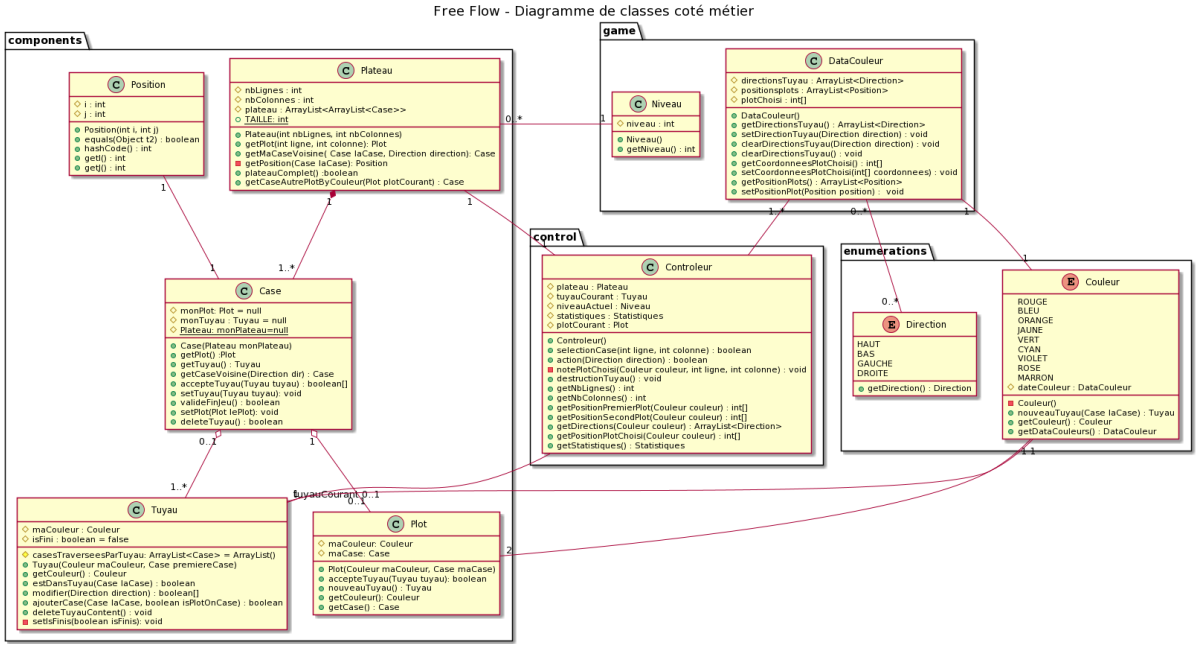
Free Flow - Diagramme de Séquence - Scénario Tentative Chevauchement Tuyau 2 Sur Tuyau 1(vue utilisateur)



03. CONCEPTION

A. COTE METIER

Pour commencer cette partie, on présente ci-dessous le diagramme de classe de notre application (coté machine) afin de pouvoir comprendre plus facilement les éléments en jeu.



En ayant repris en grande partie le modèle développé lors du TD, nous allons développer uniquement les éléments que nous avons rajoutés et qui ne sont pas explicites.

- La classe Controleur:
  - Elle possède le niveau de la partie en cours comme attribut
  - Elle possède un objet capable de faire des statistiques sur la partie en cours en attribut
  - Elle possède le plot qui a engendré le tuyau courant en attribut
  - Elle possède la méthode notePlotChoisi(Couleur couleur, int ligne, int colonne) : void qui définit les coordonnées d'un plot d'une couleur donnée
  - Elle possède la méthode getPositionPremierPlot(Couleur couleur) : int[] qui permet de récupérer les coordonnées du premier plot d'une couleur
  - Elle possède la méthode getPositionSecondPlot(Couleur couleur) : int[] qui permet de récupérer les coordonnées du second plot d'une couleur

- La classe Case :
  - Elle possède le plateau qui la contient comme attribut
  - Elle possède la méthode accepteTuyau(Tuyau tuyau) :boolean[] qui renvoit deux booléens, un pour dire que on a rajouté une case à un tuyau et un pour dire si le tuyau est fini.
  - Elle possède la méthode deleteTuyau() : boolean qui supprime le tuyau qui a voulu se mettre sur la case ou qui était présent sur la case et retourne si ce tuyau était terminé afin de pouvoir faire les statistiques sur le nombre de tuyaux terminés

- La classe Plot :
  - Il possède des attributs qui indiquent sa couleur et la case sur laquelle il se situe
  - Il possède une méthode accepteTuyau(Tuyau tuyau) : boolean qui lui permet de dire à un tuyau s'il peut se raccorder à ce plot.

- La classe Tuyau
  - Un tuyau possède un attribut isFini qui lui permet de savoir qu'il a ses deux bords qui touchent un plot et donc qu'il ne doit pas pouvoir continuer à s'étendre
  - Il possède aussi un attribut casesTraverseesParTuyau qui est un ArrayList de case qui contient l'ensemble des cases sur lesquelles est le tuyau.
  - Il possède une méthode estDansTuyau(Case laCase) : boolean qui lui permet de dire à une case si elle figure parmi les cases qu'il traverse.
  - Il possède une méthode modifier(Direction direction) : boolean[] qui retourne un tableau de boolean. Le premier correspondant au fait que le mouvement est possible et le deuxième correspondant au fait que le tuyau doit se détruire car l'utilisateur a fait une action « interdite »(ex : vouloir couper un autre tuyau ou rentrer dans un plot de la mauvaise couleur) et le troisième correspondant au fait que le tuyau était fini afin de modifier les statistiques de jeu
  - Il possède une méthode ajouterCase(Case laCase, boolean isPlotOnCase) : boolean qui ajoute la case au tuyau et retourne si le tuyau viens d'être finis ou non
  - Il possède une méthode deleteTuyauContent() : void qui lui permet de vider son arrayList de cases traversées si on lui demande de se détruire.

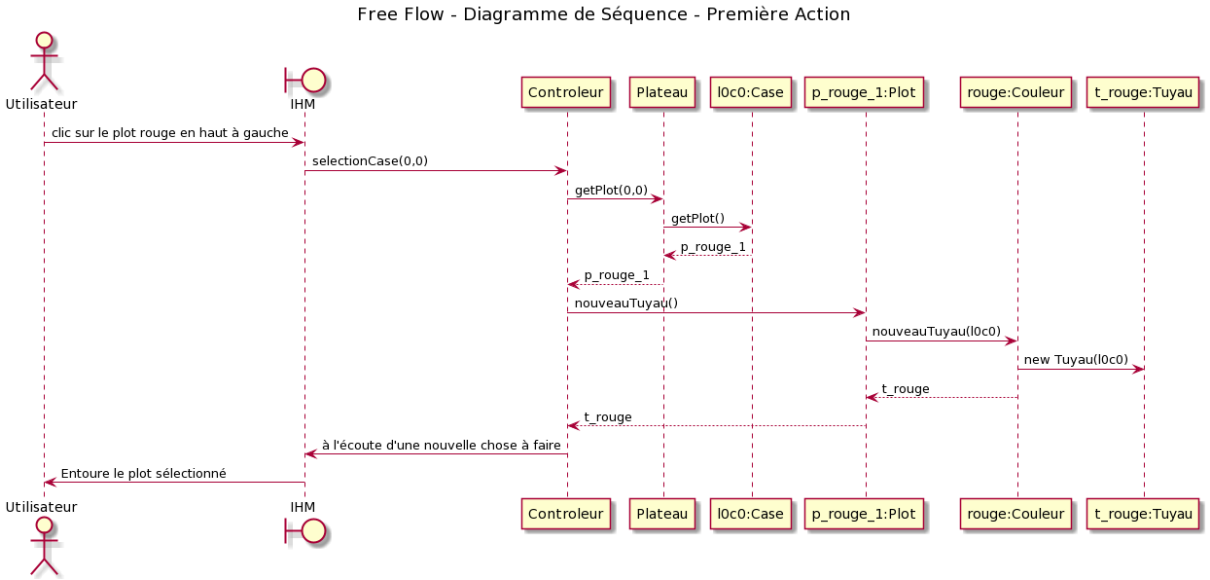
- La classe Position :
  - Il s'agit d'une classe qui permet de repérer les cases sur le plateau
  - Elle possède deux attributs : une ligne i et une colonne j
  - Elle permet surtout de pouvoir redéfinir la méthode equals(Object t2) : boolean utile pour faire de la comparaison sur notre ArrayList de Plateau contenant les cases.

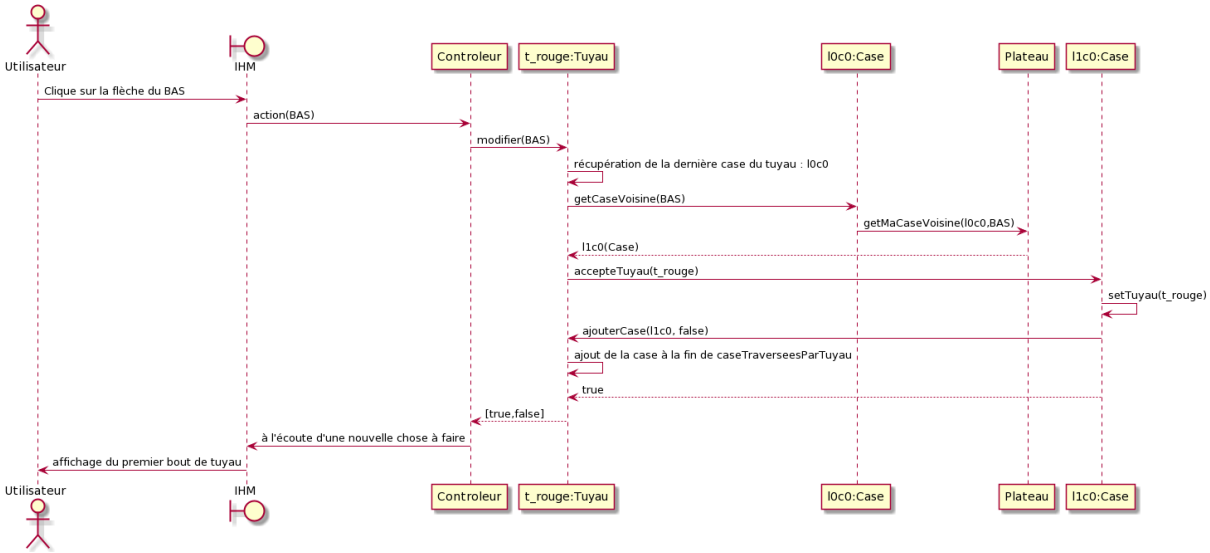
- La classe DataCouleur :
  - Elle possède un ArrayList de directions directionsTuyau qui correspond aux directions prises par un tuyau d'une certaine couleur
  - Elle possède un ArrayList de positions positionsPlots qui correspond aux positions des deux plots d'une certaine couleur
  - Elle possède un tableau de int plotChoisi qui correspond aux coordonnées du plot sélectionné et qui est à l'origine du tuyau qui se construit

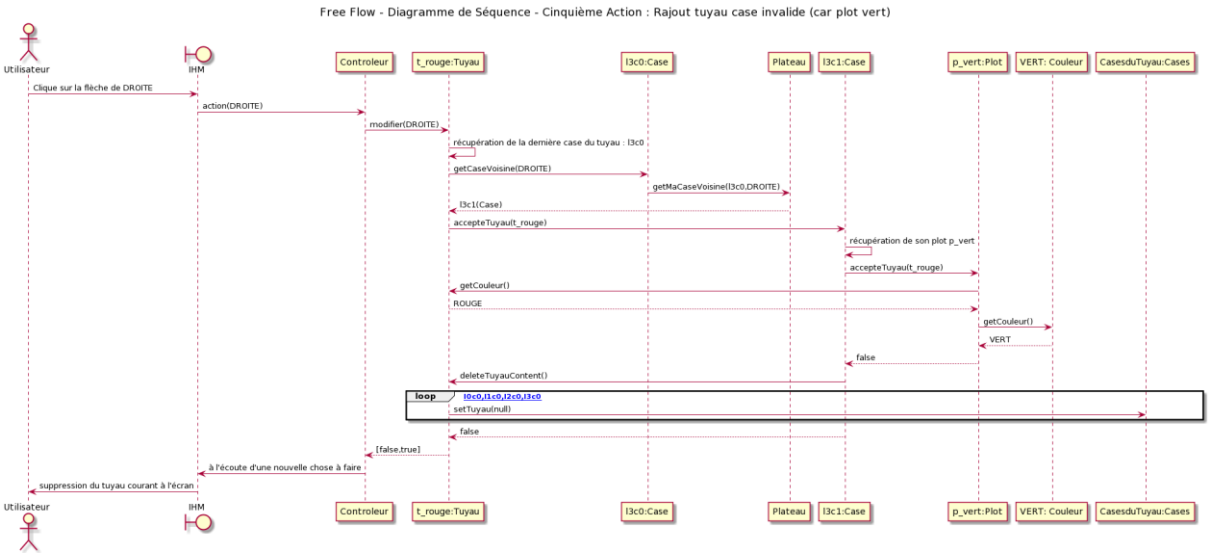
- L'énumération Couleur :
  - Elle possède un attribut dataCouleur qui permet de gérer les éléments associés à une couleur (directions du tuyau et positions des 2 plots)

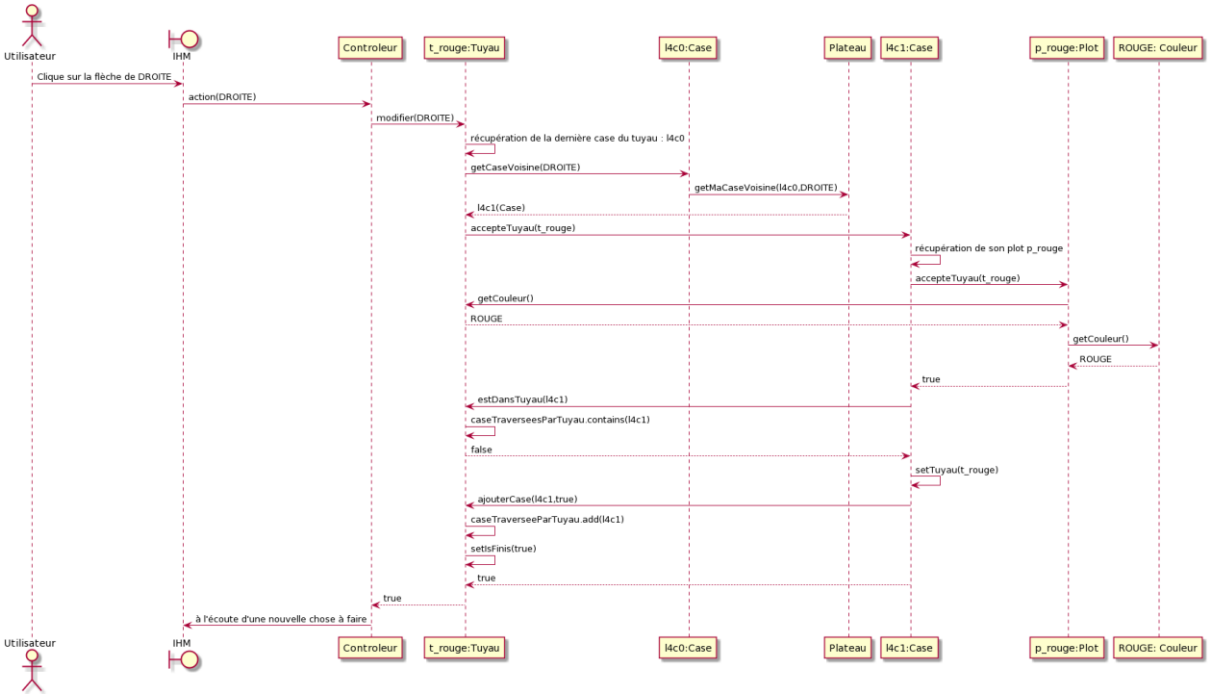
- La classe Plateau :
  - Le plateau contient un ArrayList 2D des cases. Ce plateau est créé grâce aux attributs nbColonnes et nbLignes pour ses dimensions. Pour chaque case, on verifie pour chaque couleur si elle doit accueillir un Plot, et on fixe l'attribut monPlot de la case au Plot correspondant le cas échéant. Les coordonnées des Plots sont récupérées via l'objet dataCouleur.
  - Il possède aussi une constante TAILLE qui indique sa taille afin de pouvoir redéfinir hashCode() :int dans Position().
  - Le plateau possède une méthode getPosition(Case laCase) :Position qui permet de renvoyer la position d'une case connaissant cette case.
  - Le plateau possède une méthode getPlot (int ligne, int colonne) : Plot qui permet d'aller chercher le plot sur une case à une certaine position via la méthode getPlot de la classe Case.
  - Il possède aussi la méthode getCaseAutrePlotByCouleur(Plot plotCourant) : Case qui retourne la case sur laquelle est située l'autre plot de la même couleur que plotCourant .

Afin d'illustrer les utilisations de ces différentes classes, on propose les séquences suivantes expliquant le comportement du système vis-à-vis d'actions de l'utilisateur :



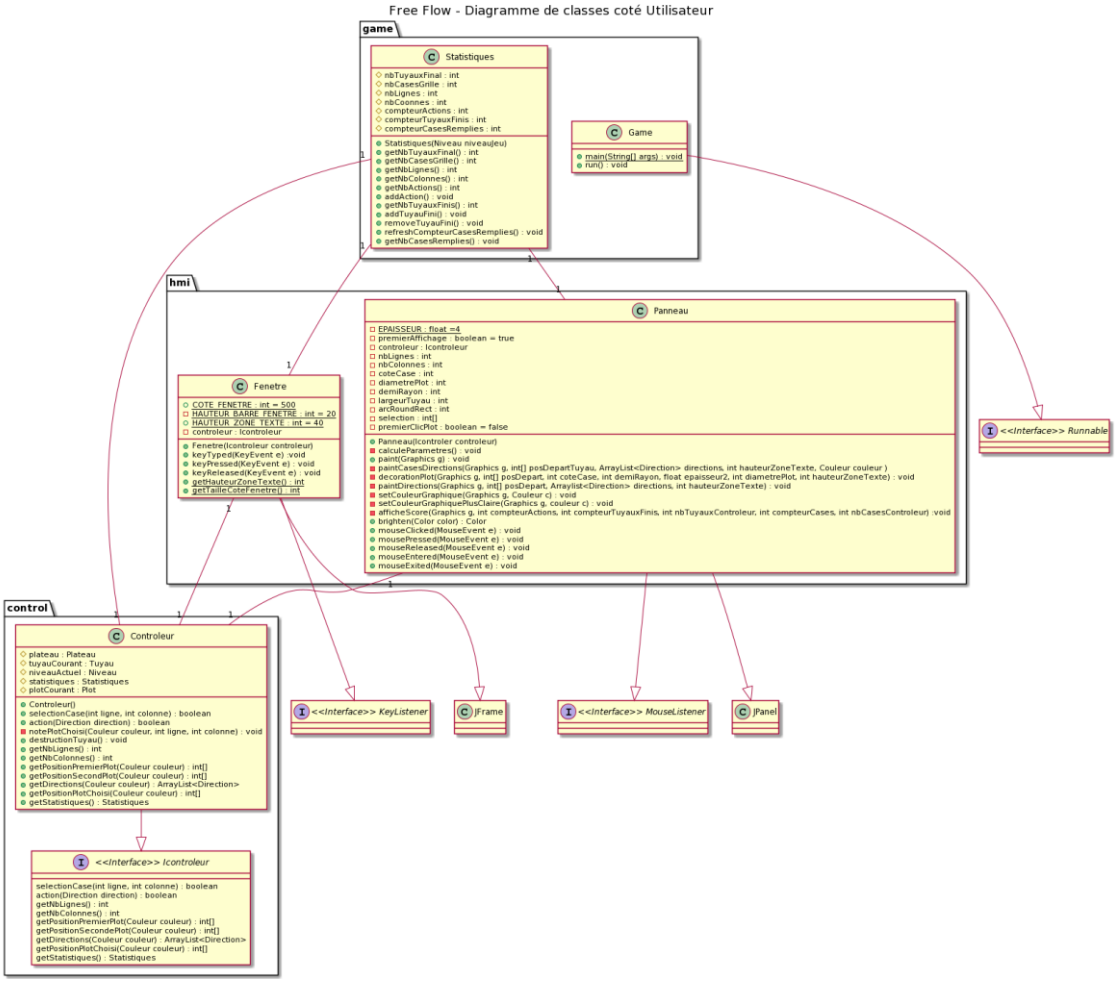



Free Flow - Diagramme de Séquence - Cinquième Action : Rajout tuyau case invalide (car plot vert)
Utilisateur
IHM
Controleur
t_rouge:Tuyau
l3c0:Case
Plateau
l3c1:Case
p_vert:Plot
VERT: Couleur
CasesduTuyau:Cases
Clique sur la flèche de DROITE
action(DROITE)
modifier(DROITE)
récupération de la dernière case du tuyau : l3c0
getCaseVoisine(DROITE)
getMaCaseVoisine(l3c0,DROITE)
l3c1(Case)
accepteTuyau(t_rouge)
récupération de son plot p_vert
accepteTuyau(t_rouge)
getCouleur()
ROUGE
getCouleur()
VERT
false
deleteTuyauContent()
loop
l0c0,l1c0,l2c0,l3c0
setTuyau(null)
false
[false,true]
à l'écoute d'une nouvelle chose à faire
suppression du tuyau courant à l'écran


Free Flow - Diagramme de Séquence - Onzième Action : connexion tuyau case valide avec Plot Rouge
Utilisateur
IHM
Controleur
t_rouge:Tuyau
l4c0:Case
Plateau
l4c1:Case
p_rouge:Plot
ROUGE: Couleur
Clique sur la flèche de DROITE
action(DROITE)
modifier(DROITE)
récupération de la dernière case du tuyau : l4c0
getCaseVoisine(DROITE)
getMaCaseVoisine(l4c0,DROITE)
l4c1(Case)
accepteTuyau(t_rouge)
récupération de son plot p_rouge
accepteTuyau(t_rouge)
getCouleur()
ROUGE
getCouleur()
ROUGE
true
estDansTuyau(l4c1)
caseTraverseesParTuyau.contains(l4c1)
false
setTuyau(t_rouge)
ajouterCase(l4c1,true)
caseTraverseeParTuyau.add(l4c1)
setIsFinis(true)
true
true
à l'écoute d'une nouvelle chose à faire

B. COTE UTILISATEUR

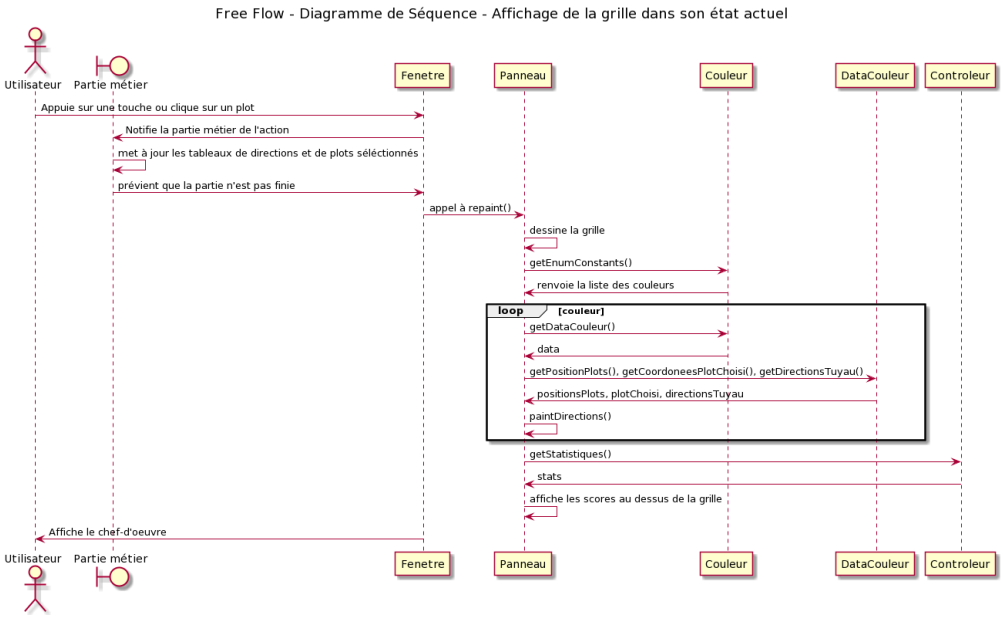
On présente désormais le diagramme de classe des objets de « l'autre côté » du Controleur

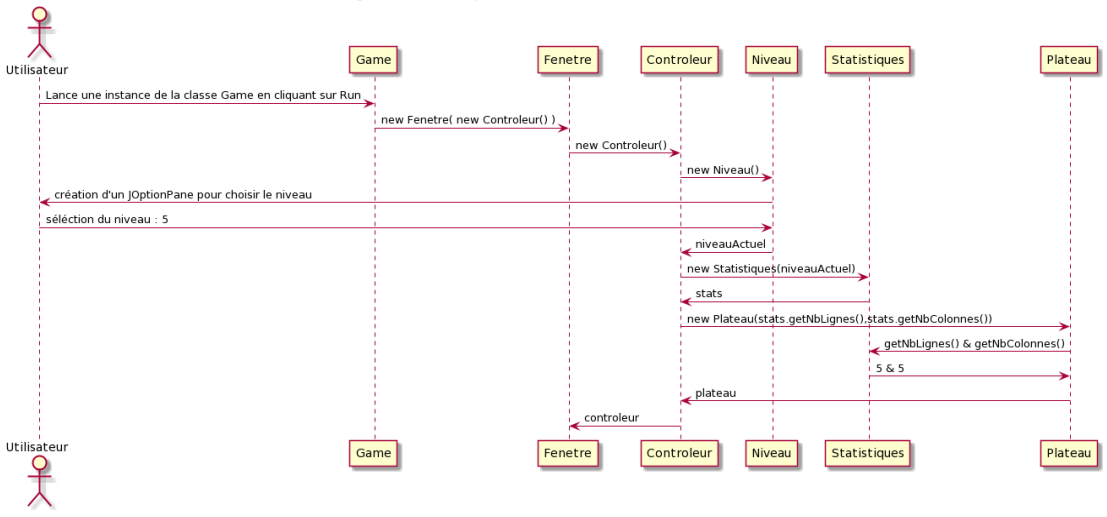


- La classe Panneau :
  - Elle possède la méthode paintCasesDirections(Graphics g, int[] posDepartTuyau, ArrayList<Direction> directions, int hauteurZoneTexte, Couleur couleur): void qui permet de peindre les cases entières en une couleur un peu plus claire que celle du tuyau lorsqu'on créé un tuyau sur une case, afin d'obtenir un effet similaire à celui sur mobile.
  - Elle possède la méthode afficheScores(Graphics g , int compteurActions , int compteurTuyauxFinis , int nbTuyauxControleur , int compteurCases , int nbCasesControleur) : void qui permet de tenir l'utilisateur au courant de son avancée dans le jeu en affichant des statistiques au-dessus de la grille de jeu.
  - Elle possède la méthode brighten(Color color) : Color où Color est la classe de java.awt.Color et qui permet de rendre une couleur plus claire

- La classe Statistique :
  - Elle permet de consulter les statistiques de la partie en cours en temps réel.
  - Elle possède des attributs qui donnent le nombre de tuyau à créer pour finir le niveau, le nombre de cases sur la grille du niveau considéré, le nombre de ligne et de colonnes du niveau considéré.
  - Elle possède un attribut compteurActions qui compte le nombre d'actions réalisées par le joueur lors de la partie.
  - Elle possède un attribut compteurTuyauxFinis qui compte le nombre de tuyaux que le joueur a réussi à construire.
  - Elle possède un attribut compteurCasesRemplies qui compte le nombre de cases qui sont recouvertes d'un tuyau et/ou d'un plot.

Afin d'illustrer les utilisations de ces différentes classes, on propose les séquences suivantes expliquant le comportement du système vis-à-vis d'actions de l'utilisateur :





04. REALISATION

Pour coder cette application nous avons décidé de nous repartir les taches de la manière suivante :

Hugo s'est occupé de la partie interface graphique et contrôleur

Vincent s'est occupé de la partie métier appart le contrôleur

Nous avons ainsi pu avancer en parallèle sur chacune des parties sans trop dépendre de l'autre. En effet Hugo a pu coder son interface graphique en partant de la base donnée dans le sujet du projet. Ainsi il avait la possibilité de créer des phases de jeu artificiellement sans avoir réellement besoin des objets métiers. De la même manière, Vincent a pu coder le coté métier de l'application sans les actions de l'utilisateur (utilisation d'une classe Main où on entrait une suite d'action de l'utilisateur via des appels de méthodes et qui se déroulait alors séquentiellement lors du run, on affichait l'état du plateau au cours du temps avec la méthode toString() : String du plateau laissée dans le code dans un but d'explication). Nous pensons que cette manière de faire était vraiment intéressante car elle permettait vraiment une parallélisation des tâches et un besoin de communication uniquement pour raccorder les deux morceaux. (ce qui n'aurait pas été le cas si nous nous étions séparé l'ensemble des classes en deux aléatoirement et que nous avions dû attendre le nom d'une méthode d'une classe que l'un ne code pas pour que l'autre puisse l'utiliser dans sa classe). Nous avons parlé d'un besoin de communication restreint mais nous tenons à mentionner que par le fait que nous sommes dans la même collocation, nous avons pu travailler ensemble et nous avons en réalité décidé de ne faire que des sessions de code communes pour pouvoir avancer plus vite en s'aidant mutuellement sur des points de blocage.

Lorsque nous avons tous les deux finis nos parties, nous avons dû recoller ces deux morceaux, via les différentes fonctions appelées par le Controleur. Cette étape n'a pas été très compliquée grâce à Eclipse et, nous l'espérons, des bonnes pratiques de code. Nous avons alors pu tester des cas non envisagés par l'un ou l'autre. Il est vrai que lorsque l'on code une partie, on a tendance à toujours exécuter les mêmes tests et on ne s'imagine pas forcément une action particulière que pourrait faire l'utilisateur.

Par exemple, lors de la phase de conception du modèle métier, Vincent avait oublié de prévoir le cas où l'utilisateur décidait de créer un tuyau et de le faire boucler sur le plot de départ. C'est Hugo qui a eu cette idée (Hugo a eu la plupart des idées bizarres que les utilisateurs pourraient avoir en fait, il faut l'admettre)

Nous sommes alors arrivés à une phase itérative du projet ou l'une ou l'autre des parties rencontrait un problème et cherchait sa source via un système de print dans la plupart des méthodes appelées qui permettait de voir les appels de méthodes au cours du temps. Nous résolvions cette erreur et nous remettions à tester l'application sur un autre cas limite afin de la rende la plus robuste possible.

La partie dont nous venons de parler concernait la base de l'application (une application qui fonctionne mais sans beaucoup d'innovations). Nous avons ensuite voulu nous rapprocher le plus possible du vrai jeu, en ajoutant par exemple les différentes statistiques ou le fond de case

qui change de couleurs lorsqu'un tuyau passe dessus. Cela nous a pris un peu de temps car il a fallu remodeler certaines de nos classes pour ajouter ces éléments.

Une fois l'esthétique convenable, nous avons décidé d'offrir la possibilité de pouvoir changer de niveau. Cette partie a été plus complexe car il nous a fallu réfléchir aux modifications à apporter pour avoir un nombre de cases variables. Nous avons ensuite pu coder l'interface graphique qui permettait le choix du nombre de cases sur le plateau.

Liste des tests mis en place :

- Partie métier hors Controleur :
  - Tests unitaires sur nos différentes méthodes avec un système de print et une réflexion sur les valeurs attendues lors de nos appels
  - Tests de chaine via l'analyse des séquences d'appel qui sont réalisées dans le jeu et grâce à un système de print
  - Test de scénario via la clase Main qui permettait de simuler des cas d'utilisation précis
  - Test d'intégration via l'interface graphique, correction des cas non pris en compte (Monkey Testing) en réutilisant le test de scénarios puis en réessayant depuis l'interface graphique
- Partie IHM + Controleur :
  - Tests unitaires et de chaine sur nos différentes méthodes grâce à l'interface graphique et au controleurBouchon
  - Test d'intégrations réalisés directement sans passer par la case tests de scénario car cela nous semblait plus judicieux de tester directement en reliant à la partie métier

- Tests utilisateurs : nous avons fait tester notre jeu à nos 4 autres colocataires

05. Conclusion

Si nous avons commencé ce projet plutôt bien avance, nous avions tout de même probablement sous-estimé le temps nécessaire pour produire un résultat satisfaisant. Nous sommes cependant très contents d'y avoir consacré une bonne partie de nos dernières journées, nous sommes assez fiers du résultat et espérons que le rendu ainsi que le code vous plairont.

Nous faisons partie de l'équipe informatique du BDS et à ce titre avons déjà codé de longues heures ensemble dans un climat bien plus stressant. C'est pourquoi il était très agréable de pouvoir se consacrer sereinement à ce projet, qui a eu le double avantage de nous amuser mais surtout de fixer solidement les enseignements vus en cours. Nous pensons que l'implémentation de ce jeu a été une excellente expérience et qu'elle nous permettra de nous souvenir encore plus aisément de la syntaxe en Java et de l'approche orienté objet.

Au cours du projet, il était très intéressant de découvrir d'où provenaient les erreurs que nous rencontrions. Nous avons par exemple longtemps cru que notre code trop lourd ne permettait qu'une fréquence de frappes sur le clavier en deçà d'un certain seuil relativement bas, avant de découvrir une ultime erreur au cours de nos tests et d'enfin comprendre ce problème de vitesse. Nous avons pu ensuite bien nous amuser (surtout Hugo) à tester la réaction du jeu à plus de 10 frappes par secondes.

Etre capable de montrer ce produit fini a été aussi une grande source de satisfaction. Nos colocataires ont pu tester notre rendu final et leurs retours ont valorisé notre effort.

Pour ce qui est des améliorations, nous aurions si nous avions eu plus de temps commencé par implémenter le cliqué déplacé pour créer les tuyaux, et nous aurions ajouté la coupure de l'ancien par le nouveau tuyau ainsi que la possibilité de récupérer un tuyau non terminé là où on l'a laissé, cruciale pour les niveaux complexes en 10x10. Nous aurions pu aussi étayer notre base de données de niveau, voir construire un algorithme de génération automatique de niveaux.

Enfin, nous souhaitions remercier l'équipe en charge de cet électif de nous avoir donné ce projet et d'en avoir fait un défi appréhendable qui a su occuper nos heures de couvre-feu.

Vincent & Hugo.

06. Annexe

Code pour le diagramme des cas d'utilisation :

diag_usecase.txt

Code pour le diagramme de séquence des actions utilisateur

diag_sequence_useractions.txt

Code pour le diagramme de séquence scénario « parfait »

diag_sequence_perfectcase.txt

Code pour le diagramme de séquence scénario « erreur »

diag_sequence_errorcase.txt

Code pour les diagrammes de classes :

diag_classe_cote_metier.txt

diag_classe_homme.txt

Code pour les différents diagrammes de séquences de la partie conception :

Diag_sequence_NUMEROACTION.txt

Code pour le diagramme d'affichage de la grille de jeu dans son état actuel :

Diag_sequence_affichage_grille.txt

Code pour le diagramme de choix du niveau et de création du Controleur :

Diag_sequence_niveau_controleur.txt