CentraleSupélec

SG6

Génie logiciel orienté objet

Rapport

Projet "Numberlink"

Élève :
Mohsine Zireg

Enseignant :
Dominique Marcadet

30 janvier 2022

Table des matières

1 Introduction **2**

2 Cahier des charges **2**

3 Expression du besoin **2**
3.1 Initialisation du jeu . . . 2
3.2 Début du jeu . . . 3
3.3 Construction des chemins . . . 3
3.4 Réinitialisation de la grille . . . 3
3.5 Fin du jeu . . . 3
3.6 Communication Jeu-Utilisateur . . . 3

4 Conception **4**
4.1 Diagramme de classes du modèle métier . . . 4
4.2 Description des classes et leurs attributs . . . 4
4.2.1 NumberLinkWindow . . . 4
4.2.2 NumberLinkPanel . . . 4
4.2.3 ControllerStub . . . 4
4.2.4 IController . . . 5
4.2.5 Direction . . . 5
4.2.6 Grid . . . 5
4.2.7 Cell . . . 5
4.2.8 Path . . . 5
4.2.9 Tag . . . 5
4.2.10 End . . . 5
4.3 Diagramme de séquences . . . 6
4.3.1 Clic sur un point . . . 6
4.3.2 Action par flèche du clavier . . . 6
4.3.3 Fin du jeu . . . 7
4.4 Autres méthodes non mentionnées sur les diagrammes . . . 7

5 Réalisation **8**
5.1 Chronologie . . . 8
5.2 Réalisation du code à partir du diagramme UML . . . 8
5.3 Tests réalisés . . . 8
5.3.1 Tests unitaires . . . 8
5.3.2 Tests boite blanche . . . 8

6 Conclusion et pistes d'amélioration **9**
6.1 Pistes d'amélioration . . . 9

1 Introduction

Le projet réalisé est un jeu assez amusant qui challenge l'intelligence et organisation d'esprit du joueur. Ce dernier doit relier des points de même couleur dans une grille tout en parcourant les différentes cases de cette grille. La réalisation de ce projet va etre explique ci dessous suivant différentes sections. Après l'introduction, on passera au Cahier de charges, l'objectif général du jeu. Puis l'expression des besoins, une partie détaillée où on pourra voir les attentes d'un joueur du jeu de son point de vue. Et la conception où il faudra expliquer de façon détaillée le fonctionnement du code derrière les coulisses avec les différents package et différentes classes et scénarios sous forme de diagramme de séquences. Ensuite, on passera à la réalisation, en d'autres termes, le passage du modèle métier UML au code Java avec les différentes classes et différents test réalisés pour s'assurer du bon fonctionnement du jeu. On finira avec une conclusion des différentes fonctionnalités prévus pour améliorer le jeu et dont je n'ai pas eu assez de temps pour les implémenter, mais ces dernières vont être bien expliquées surtout en matière de choix de conception.

2 Cahier des charges

Le jeu Numberlink doit permettre au joueur d'essayer de relier 2 points distants de la grille. Il devra aussi s'assurer que lorsque le joueur aura relié tous les 2 points de la grille, il aura aussi rempli les différentes cases. En d'autres termes, chaque case de la grille doit contenir un point ou un chemin. La taille de la grille, le nombre d'étiquettes *(Chaque 2 points liés correspondent à une étiquette/couleur)* et la position des différents points peuvent être entrés par l'utilisateur. Les détails des différents éléments du jeu de point de vue interne et externe vont être expliques dans les parties ci dessous.

3 Expression du besoin

3.1 Initialisation du jeu

De point de vue utilisateur, ce dernier voudra avoir une interface graphique qui s'affiche une fois le jeu démarré. Si le joueur veut que le jeu lui propose une grille prédéfinie, il n'aura qu'à répondre par "Oui" à la question posée au début "Voulez-vous jouer avec une grille prédéfinie ?". Toute autre réponse *(Comme non)* mènera le joueur à choisir lui même les nombre d'étiquette *(Ceci correspond à la difficulté du jeu. En effet, plus d'étiquettes mène à plus de difficulté)*, le nombre de lignes et colonnes de la grille et enfin la position suivant les 2 axes vertical et horizontal de chaque 2 point constituant une étiquette.

3.2 Début du jeu

Après ces différents choix, le jeu commence. Le joueur aura donc à réaliser un clic gauche avec la souris sur les points qu'il souhaite commencer à relier. Il pourra commencer par n'importe quelle étiquette/couleur, ou bien par n'importe quelle point des 2 points correspondant à une étiquette. Seulement, après avoir commencer un chemin à partir du point A d'une étiquette, il ne pourra pas cliqué sur le point B pour commencer un nouveau chemin. Si l'utilisateur veut tout de même commencer par le point B alors qu'il a déjà commencé un chemin par le point A, il pourra réinitilaiser la grille du début. La réinitilastion de la grille sera expliqué ci dessous. Le joueur veillera à cliquer sur des cases où il y'a déjà des points définis, sinon le jeu ne prendre pas en compte son clic.

3.3 Construction des chemins

L'étape qui suit le clic sur un point est la création d'un chemin à partir du point sélectionné. Le joueur essayera ainsi à arriver au 2ème point de l'étiquette, tout en veillant à finir le jeu en passant par toutes les différents cases vides. Le chemin sera construit à partir des flèches du clavier *(up,down,right,left)*. Cependant, le joueur doit veiller à ne pas passer par des cases ou des chemins ou points existent déjà. Le cas échéant, son action ne sera pas pris en compte sans avertissement par l'interface homme-machine puisque c'est évident qu'il ne faut pas passer par des cases déjà utilisées. Par contre, si le joueur, par son action sur les flèches du clavier, dépasse les limites de la grille, le jeu ne prendra pas en compte cette dernière en émettant un avertissement au joueur. A part ces cas extrêmes, le joueur pourra construire son chemin comme il veut, et pourra aussi réinistilaiser la grille *(Effacer tout les chemins)* si il lui semble qu'il a effectué une erreur sur un chemin.

3.4 Réinitialisation de la grille

Un joueur peut souhaiter redémarrer le jeu du début pour différentes raisons citées ci-dessus *(Nouveau choix du point de départ, Chemin mal construit..etc)*. Pour faire cela, il n'aura qu'à effectuer un clic droit n'importe où sur la grille. Ceci relancera le jeu du début. Cette action peut être Réalisé autant de fois que le joueur voudra et affichera à l'utilisateur à chaque fois un message "Grid reset !".

3.5 Fin du jeu

Le jeu se terminera lorsque le joueur aura réussi à relier tout les points des étiquettes en parcourant toutes les cases vides de la grille. Un message sera ainsi affiché lui annonçant son génie et sa victoire !

3.6 Communication Jeu-Utilisateur

Le jeu communiquera à l'utilisateur la case où il a cliqué, les flèches actionnées et quelques messages d'avertissement mentionnés dans les section ci-dessus.

4 Conception

4.1 Diagramme de classes du modèle métier

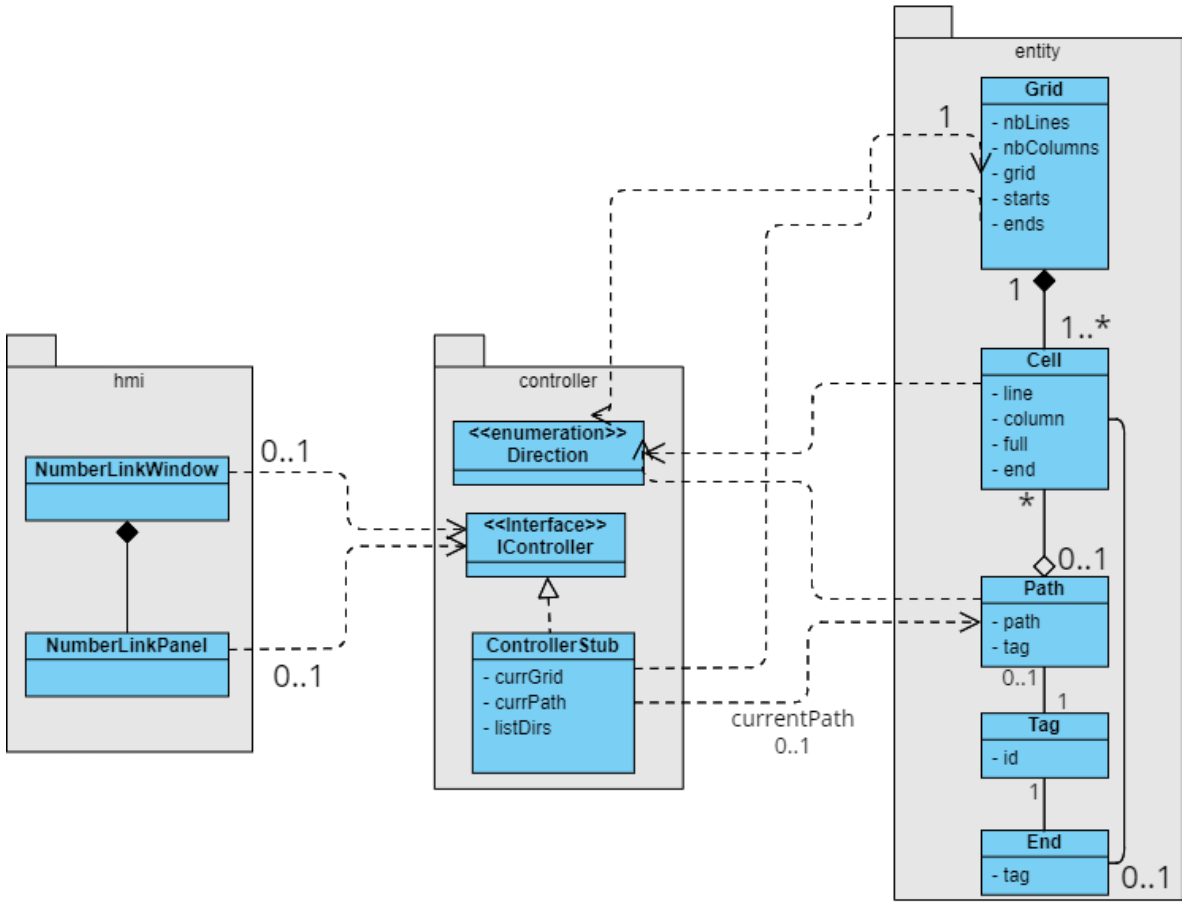

FIGURE 1 – Diagramme UML

4.2 Description des classes et leurs attributs

4.2.1 NumberLinkWindow

La classe NumberLinkWindow implémente la fenêtre à afficher et le module KeyListener qui permet de capter les appuis sur les flèches du clavier.

4.2.2 NumberLinkPanel

La classe NumberLinkPanel implémente l'afficheur à utiliser et le module MouseListener qui permet de capter les appuis sur la souris.

4.2.3 ControllerStub

La classe qui fait office de contrôleur sur la base du patron MVC. Elle contient la grille actuelle currGrid instance de Grid, et le chemin en cours de construction currentPath instance de Path.

4.2.4 IController

L'interface du contrôleur qui définit les fonctions à implémenter.

4.2.5 Direction

Une classe qui définit les différentes valeurs possibles de Direction *(up,down,right,left)*

4.2.6 Grid

La classe qui s'occupe de générer la grille avec comme paramètre le nombre de lignes, de colonnes, d'étiquettes et des points des différentes étiquettes.

4.2.7 Cell

La classe qui représente une case de la grille, définie par sa ligne, sa colonne, si elle contient ou pas un chemin et si elle contient ou pas un point d'une étiquette *(full)*

4.2.8 Path

La classe qui représente les différents chemins construits *(path)* sous forme de ArrayList<Cell> et les tags/étiquettes correspondants.

4.2.9 Tag

La classe qui mémorise les différentes étiquettes avec leur id.

4.2.10 End

La classe qui représente les différents points ou ends.

4.3 Diagramme de séquences

4.3.1 Clic sur un point

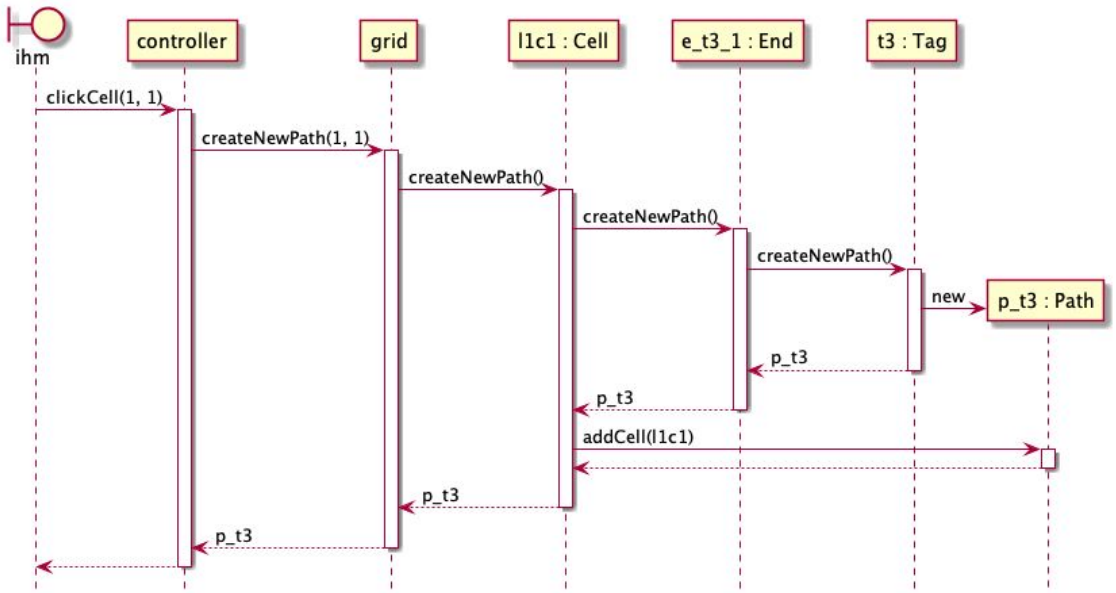

FIGURE 2 – Diagramme de séquences

4.3.2 Action par flèche du clavier

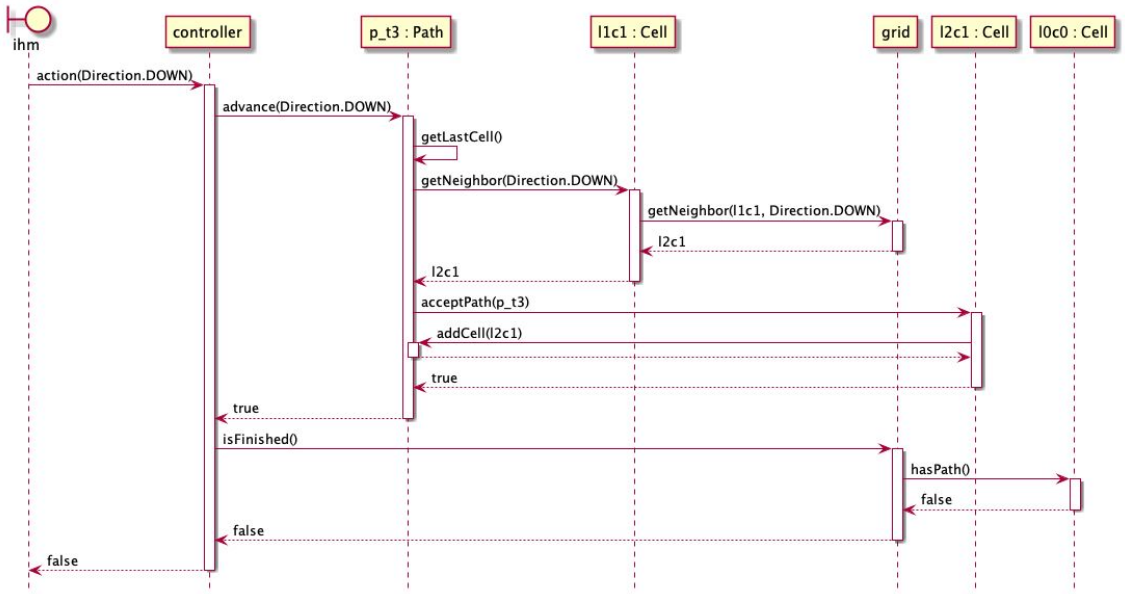

FIGURE 3 – Diagramme de séquences

4.3.3 Fin du jeu

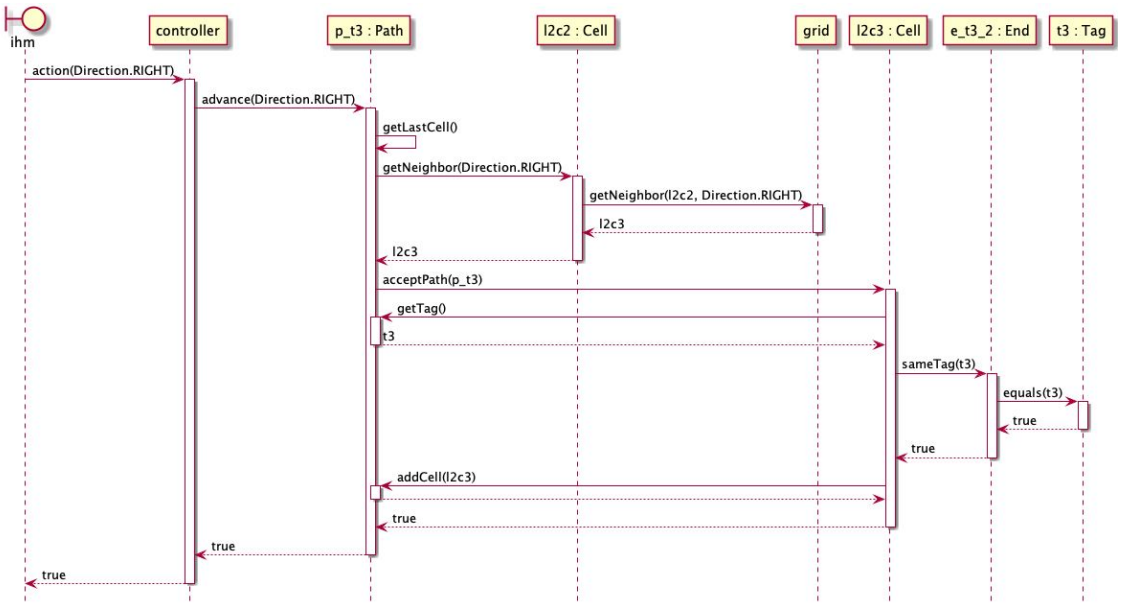

FIGURE 4 – Diagramme de séquences

4.4 Autres méthodes non mentionnées sur les diagrammes

Dans presque toutes les classes, des getters et setters nécessaires ont été implémentés.

Path

- La méthode getTagId a été ajouté pour remplacer getTag.getId afin de respecter la loi de Demeter.

Grid

- La méthode swapStartEnd est utilisée afin d'enverser les points A et B d'une seule étiquette et permettre au joueur de commencer de n'importe quel point. - La méthode isStart est utilisée pour détecter si le point sélectionné par le joueur appartient à la liste des points de départ des étiquettes.

ControllerStub

- La méthode resteGrid réinitialise la grille à son état de départ par demande du joueur. - La méthode setListDirs prépare la liste des directions pour l'utilisation de la part de l'hmi.

NumberLinkPanel

- On a ajouté à la méthode mouseClicked une détection du clic droit pour réinitialise la grille.

5 Réalisation

5.1 Chronologie

Ce travail est individuel et par conséquent je n'ai pas pu implémenter toutes les fonctionnalités que je j'avais en tête et mentionnées ci-dessous en conclusion. J'ai réalisé la base du projet, c'est à dire un jeu fonctionnel dans la semaine avant les partiels. Le reste qui consiste de la révision du code, la réalisation de test, l'implémentation de nouvelles fonctionnalités comme la réinitialisation de la grille et d'autres a été réalisé pendant et après les partiels.

5.2 Réalisation du code à partir du diagramme UML

J'ai choisi de créer une instance de Grid nommée currGrid et qui servira de grille pendant tout le jeu. Une variable currPath représente aussi le chemin en cours de construction. La liste de direction a été aussi remplacé par un ArrayList<ArrayList<Direction» afin de faciliter l'ajout de nouvelles directions à la grille.

En se qui concerne les différentes associations sur le diagramme de classe UML, j'ai opté quelques fois à implémenter la classe comme attribut de la classe mère. D'autres fois, il fallait juste implémenter un nouvel attribut qui faisait référence à la classe en question *(Comme Tag tag dans la classe End)*. D'une autre part, j'ai opté pour des attributs statiques pour la classe Grid afin de faciliter l'obtention de quelques attributs par les autres classes *(Comme la classe Tag)*. Ce choix reste justifié tant qu'on travaille avec une seul grille par jeu et du coup les attributs sont constants et lié à la classe elle même et pas une instance dans ce cas d'utilisation.

5.3 Tests réalisés

5.3.1 Tests unitaires

Des tests unitaire ont été implémentés pour tester le bon fonctionnement de la génération des différentes entités. Elle se trouve dans la classe EntityTest dans le package entity.

5.3.2 Tests boite blanche

J'ai effectué des tests boite blanches manuellement vu que je connais parfaitement le code que j'ai conçu. Par conséquent, je l'ai testé en passant par des cas extremes ; Sortie de la grille, le joueur essaie de passer par une case déjà occupé par un chemin ou un point d'une étiquette, le joueur change d'avis et souhaite commencer par un nouveau point, et enfin le joueur décide de ressayer le jeu plusieurs fois. Et dans tous les cas précédents, le code a donné un comportement exemplaire.

6 Conclusion et pistes d'amélioration

En résumé, ce projet peut avoir l'air au début pas très intéressant puisque c'est un jeu assez basique. Mais il me semble très important dans la mesure où il m'a aidé à bien implémenter les différents principes vu en cours et qui restaient un peu flous sans application directe comme celle-ci. C'est notamment le cas des patrons MVC, des règles de conception, de normes de qualité, des interfaces et interfaces graphique homme-machine.

6.1 Pistes d'amélioration

Plusieurs pistes d'améliorations sont possibles et facilement implémentables. J'aurai pu les ajouter si ce n'était la contrainte de temps et partiels.

Génération automatique de grilles

On peut aussi générer automatiquement les différentes grilles du jeu de façon aléatoire, ou bien sur la base d'un solveur implémenté indépendamment qui vérfifiera s'il existe une solution pour les grilles générés aléatoirement et donnera au joueur une des grilles qui aura une solution.

Plus de niveaux..

On peut commencer en ajoutant différents niveaux du jeu qui augmentent de difficultés pour chaque niveau.

Want a "hint" ?

L'idée est d'utiliser le solveur afin de donner au joueur des indications sur la solution.

I give up :(

C'est dans le titre.. On affichera au joueur la solution trouvée par le solveur automatique.

Fin