ED3-C Année 2023-2024

RAPPORT DE PROJET

Crazy Plumber

Réalisé par
Félix MARTINS-DUCASSE
Léo LIN
Yanis MANSOURI
Julie LE ROY
Natacha WENG

Université Paris Cité

SOMMAIRE

SOMMAIRE.....2
I. INTRODUCTION.....3
II. OBJECTIFS.....4
A. Consignes initiales.....4
B. Contraintes et Guidelines.....4
III. CONCEPTION DU PROJET.....5
A. Cahier des charges.....5
1. Charges minimales :.....5
2. Extensions :.....5
B. Logique du jeu.....5
C. Répartition du travail.....5
IV. ASPECT FONCTIONNEL.....6
A. Structure.....6
B. Extensions.....7
1. Modes de jeu.....8
2. Implémentation des indices.....8
3. Sauvegarde.....8
4. Succès déblocable.....9
C. Implémentations GUI.....9
1. Interfaces.....9
2. Animations.....10
V. PROBLÈMES RENCONTRÉS.....11
1. Problème animation d'eau.....11
VI. PISTES D'AMÉLIORATION ENVISAGEABLE.....12
VII. BILAN.....13

I. INTRODUCTION

Dans le cadre de notre quatrième semestre de licence d'informatique générale, il nous a été demandé de réaliser un jeu parmi une liste de projet. Nous devions ainsi réaliser un jeu dans le style Crazy Plumber en groupe de 5, mettant à l'épreuve nos connaissances en java, en POO et en organisation autour d'un projet.

II. OBJECTIFS

A. Consignes initiales

Réaliser un jeu de plombier, c’est à dire un jeu vidéo de réflexion dans lequel le joueur incarne un plombier qui doit connecter des tuyaux pour éviter une fuite et amener l’eau d’une ou plusieurs entrées à une ou plusieurs sorties

B. Contraintes et Guidelines

Réaliser le jeu en implémentant plusieurs niveaux différents et en utilisant la librairie java swing pour l’interface graphique. Avec une réunion entre tous les membres du groupe et notre professeur référent chaque semaine nous pouvions suivre l'avancée de notre projet et se fixer des objectifs réalisables.

III. CONCEPTION DU PROJET

A. Cahier des charges

1. Charges minimales :

- Logique des tuyaux
- Logique de condition de victoire
- Implémentations de plusieurs niveaux

2. Extensions :

- Système de sauvegarde
- Niveaux déblocables
- Succès déblocables
- Plusieurs modes de jeu
- Sons d'ambiance
- Plusieurs couleurs pour l'eau
- Système d'indice en jeu

B. Logique du jeu

Le jeu est composé de trois difficultés comportant 15 niveaux chacun : easy, medium et hard. Il existe trois modes de jeu, un mode normal sans contrainte, un mode Limited Move dans lequel le joueur joue selon un certain nombre de mouvements possible et un mode Timer avec un chronomètre. Des succès à débloquer sont disponibles pour le joueur.

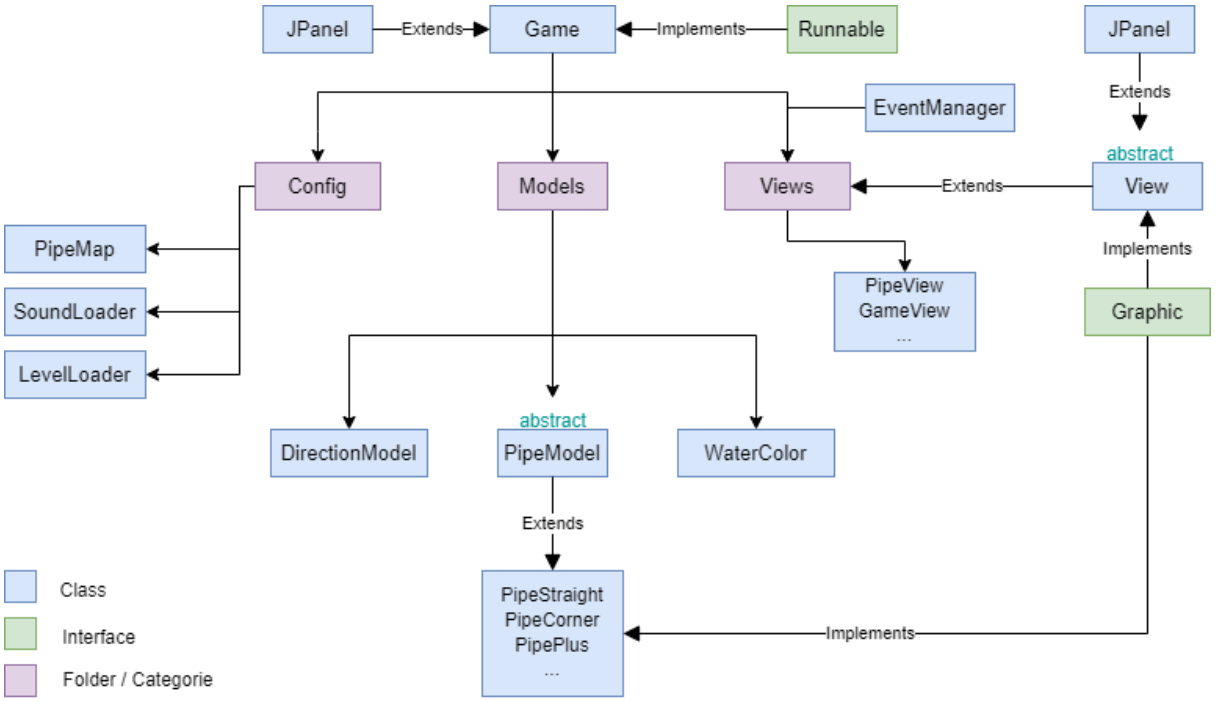
C. Répartition du travail

Chaque semaine, des tâches devaient être commencées ou à finir. Le choix des responsables de chaque issue était arbitraire. En effet, il pouvait arriver que plusieurs personnes travaillent ensemble sur une tâche spécifique qui était complexe à résoudre.

IV. ASPECT FONCTIONNEL

A. Structure

Diagram UML Simplifié du projet



(Le diagramme intègre uniquement les classes importante du code)

Pour permettre des améliorations plus simples et coder de façon plus agréable il nous fallait une structure de base polyvalente et simple d'utilisation.

Premièrement nous devons représenter les différents types de tuyaux. Naturellement une classe spécifique à chaque type de tuyaux héritant d'une classe principale PipeModel a été mise en place.

Dans un second temps nous avons dû réfléchir à l'implémentation des connexions entre les tuyaux. Étant l'interaction principale de notre projet, une implémentation pertinente, pratique et réutilisable était primordiale. Après plusieurs tentatives infructueuses, trop complexes ou peu pratiques, nous avons implémenté la solution actuelle, une table de direction.

En effet on va considérer chaque tuyaux dans son modèle de base, c'est-à-dire sans rotation appliquée, et indiquer dans la table les sorties NORD, SUD, EST, OUEST de chaque entrée NORD, SUD, EST, OUEST. Avec une bonne gestion des enum de chaque tuyaux et de chaque direction la table et les fonctions l'utilisant en demeure concise.

Quand est il des connexions lorsque les tuyaux ne sont pas dans leur modèle de base, c'est-à-dire avec une rotation ? Avec une petite gymnastique mentale et quelques schéma on se rend compte qu'une rotation inverse à a la rotation du tuyaux actuel, appliquées aux directions du modèle de base nous permet de de savoir si les tuyaux sont connectés.

Maintenant que l'on peut faire ça, l'algorithme nous permettant de savoir si notre chemin de tuyaux amène bien l'eau reste concis et naturel : une fonction récursive prenant un tuyaux et une direction en argument.

On part de chaque tuyaux start et de leur direction de sortie,

Si on trouve une arrivée alors on sait qu'il y a un chemin entre l'entrée et la sortie (attention ce n'est pas tout de suite gagné il faut gérer les fuites)

Sinon on cherche le voisin dans la direction donnée en argument,
si il est connecté à notre tuyaux actuel :
on appelle la fonction sur ce voisin et sur toutes les directions de sortie de ce dernier

Si il n'est pas connecté à notre tuyaux actuel :
on a une fuite

La victoire est donc réalisé quand il y a un chemin depuis chaque point de départ vers le point d'arriver et cela sans fuite, dans cette explication on s'abstient de quelques points de code, l'idée principale étant celle expliquée

B. Extensions

1. Modes de jeu

Pour la création des modes,nous avons créé une classe Modes qui gère tous les modes de jeu.

Le mode Limited Move fonctionne avec un tableau de taille 15 contenant le nombre maximal de mouvement possible pour les 15 niveaux, ces nombres sont adaptés selon la difficulté en cours et un entier nbMove initialisé à 0, si nbMove atteint le nombre maximal alors le joueur a perdu.

Pour le mode Timer cela fonctionne de la même manière mais avec une liste de temps et un entier remainingTime. Au début de chaque partie on démarre un timer avec TaskTimer qui étend TimerTask et s'il atteint 0 le joueur perd.

2. Implémentation des indices

L'ensemble des indices est géré dans la classe Hints. Un modèle Hint est composé d'un tuyau et d'une rotation, lorsque l'on active un indice, on change la rotation du tuyau et on fait en sorte qu'il ne puisse plus être bougé. Les indices sont récupérés à partir de fichiers textes qui sont organisés selon les différentes difficultés du jeu.

Dans un fichier, un indice correspond à un triplet d'entiers (x,y,c) où x et y sont les coordonnées du tuyau et c son coefficient de rotation. Les indices sont stockés dans une liste et lorsque le joueur appuie sur le bouton des indices, on récupère un indice s'il en reste. A l'affichage, le tuyau indice est encadré en blanc et il n'est plus possible de le tourner.

3. Sauvegarde

Pour le système de sauvegarde, la classe Player va être sérialisée pour pouvoir garder en mémoire toutes les informations importantes liées au joueur. Par exemple les niveaux qui ont été complétés ou alors les succès obtenus, ce qui va permettre au reste du programme de s'adapter par rapport aux attributs du joueur.

En outre, au lancement du jeu, l'utilisateur va pouvoir soit créer une nouvelle partie ou soit charger une partie déjà existante (désérialisation) qui a été sauvegardée auparavant.

4. Succès déblocable

Un succès est principalement composé d'une description, d'un nom et d'une condition de déblocage. L'ensemble des succès possibles du jeu sont stockés dans une liste située dans la classe Player, c'est-à-dire que chaque joueur a sa propre liste et il peut en débloquer jusqu'à huit. De plus, les succès sont représentés par des boutons sans actionListener et inactivés si le succès n'est pas encore débloqué. Ils sont mis à jour lorsque l'on arrive sur la vue des succès : pour chaque succès on vérifie sa condition de déblocage et on active le bouton si le joueur a réussi à le débloquer. Lorsque l'on passe la souris sur un succès, sa description est affichée ce qui permet au joueur de savoir ce qu'il doit faire pour pouvoir le débloquer.

C. Implémentations GUI

1. Interfaces

Pour permettre les différents affichages, nous avons fait une interface Graphic qui comporte des méthodes qui peuvent être utiles et faciliter la création de certains composants et éléments des différentes vues. Par exemple, la méthode getImage() qui renvoie l'image correspondant au chemin mis en paramètre ou encore la méthode makeButton() qui est souvent utilisée dans les vues pour créer un bouton préparé pour recevoir des icônes.

Une classe abstraite View existe dans laquelle on initialise les différentes images qu'on utilise, et définir le contenu de la fenêtre courante. Plusieurs classes qui étendent de View permettent d'afficher les différentes fenêtres voulues. Par exemple, MenuView affiche le menu, et permet de nous rediriger à l'aide de boutons vers d'autres classes qui étendent de View selon nos besoins.

Afin d'afficher la matrice des tuyaux, nous avons fait une méthode drawMap qui va parcourir la matrice de la map du jeu et faire appel à la

fonction drawPipe pour chaque tuyaux afin de les afficher. Cette méthode dessine une image de fond puis par dessus le tuyau selon sa rotation. L'image de chaque tuyau est stockée en attribut.

Auparavant pour changer de mode, le joueur devait passer par les paramètres puis mode. A présent, lorsque le joueur clique sur jouer, il est directement redirigé vers la vue avec les différents niveaux. En dessous des niveaux il y a les boutons "mode" qui permettent au joueur de changer le mode de jeu, "difficulty" pour changer la difficulté du jeu et un bouton retour. Au dessus est affiché le mode de jeu et la difficulté actuelle ce qui permet au joueur de savoir quel est le mode et la difficulté en cours.

La vue GameView correspond au panel d'une partie du jeu. Elle est composée de plusieurs boutons. Le joueur peut annuler ou remettre un mouvement s'il le souhaite, recommencer la partie ou encore récupérer des indices si jamais il bloque sur le niveau. Lorsque le joueur a fini de tourner les tuyaux, il peut appuyer sur le bouton avec la roue pour vérifier si c'est correct ou pas. Dans le cas où le chemin est correct, cela démarre l'animation de l'eau.

2. Animations

Pour l'animation de l'eau, on vérifie s' il n'y a pas de fuite dans le chemin , si il y'en a pas , on lance l'animation. L'animation se fait au niveau de PipeView on compare regarde pour chaque pipe dans le chemin si du côté où l'eau arrive l'animation est finie et on détermine le sens de l'animation à partir de cette information et on joue l'animation en 11 parties distinctes. Pour la partie de gestion des assets chaque sous classe du PipeModel charge ses propres asset d'après son Type et ainsi on charge les asset d'animation & de visuel.

Ensuite pour l'animation de rotation on a chaque clique on joue l'animation de rotation progressive grâce à une table de vérité qui vérifie qu'on a bien jouer toute les animation car sinon on risque de skip une frame sur un lag de l'ordre de la ms. Lorsque l'animation est lancée on ne peut plus relancer l'animation sans que celle-ci soit terminée.

V. PROBLÈMES RENCONTRÉS

1. Problème animation d'eau

L'animation de l'eau a été faite une première fois avec une disjonction de cas selon l'orientation et le type de pipe mais cela nous a posé un problème lors de l'animation des pipe spéciaux Double Corner et Plus . Étant donné qu'il sont composés de deux éléments distincts, les considérer comme un unique PipeModel nous empêchait de faire l'animation à travers convenablement sans avoir une incompatibilité avec l'algorithme de recherche de chemin valide pour l'eau. Alors nous avons changé et simplifié le code de l'animation et modifié les deux pipe afin qu'il ne soit pas considéré comme 1 Pipe mais deux pipe superposées ce qui nous permet de les distinguer dans l'algorithme de pathing et l'animation de l'eau.

2. Problème d'implémentation

Au début du projet, beaucoup de problèmes venaient d'un mauvais choix d'implémentation :

- Une double liste chaînée pour stocker les tuyaux et réaliser une chaîne entre l'entrée et la sortie ? Trop complexe et pas adaptée aux tuyaux comportant plusieurs sorties.
- Un tableau de direction changeant à chaque rotation ? On s'y perd rapidement et cela rend les connexions plus compliquées.

C'est donc après plusieurs échecs que nous avons mis en place notre table générale pour tous les types de tuyaux.

VI. PISTES D'AMÉLIORATION ENVISAGEABLE

Bien que le projet soit terminé, il peut être intéressant de le continuer à titre personnel. Voici les améliorations envisageables que nous n'avons pas pu implémenter :

- Nos tuyaux sont dans des cases carrées, on aurait pu généraliser à des cases représentant des polygones réguliers comme des hexagones. Pour cela il faudrait changer légèrement notre table et la matrice principale.

- Pourquoi ne pas jouer à deux ? Diviser l'écran en deux parties identiques, chacun jouant l'un après l'autre était une amélioration qu'on avait envisagée, cependant nous la trouvions peu intéressante par rapport à la sauvegarde.

- Une meilleure utilisation des couleurs que celle implémentée.

VII. BILAN

Pour conclure ce rapport, faisons un bilan de notre projet. Nous étions 5, le même groupe que le projet de Conduite de Projet du semestre dernier. Nous avons mieux exploité nos connaissances de programmation de projet en java et comme nous avons déjà travaillé ensemble l'organisation et la répartition des tâches ainsi que la résolution de problèmes était meilleure. A chaque projet nous réussissons à produire quelque chose de meilleur que la fois précédente, gagnant ainsi en expérience. Cela étant, ce projet sera surement continué personnellement afin d'arriver à un produit le plus propre et complet possible en intégrant les améliorations précédemment citées.