Université Gustave Eiffel

Patchwork

Rendu Final

Architecture et choix de programmation

Programmation en Java

RADONIAINA Gabriel
NGUYEN Hervé

15 janvier 2023

L3 Informatique

Table de matières

Introduction

Dans ce rapport nous allons expliquer nos choix jusqu'à maintenant pendant le développement pour le rendu final du jeu Patchwork dans le cadre du projet de programmation en Java.

Différence majeurs avec le rendu intermédiaire

En terme d'architecture des données originelles et représentation des éléments tel que les joueurs et patchs, il n'y a que très peu de différence.

Nous avons néanmoins, ajouté pour certaines classes comme Player quelques méthode supplémentaires qui permettent de faciliter quelques tâches dans le mode graphique.

Au niveau du mode graphique nous avons crée quelques nouvelles classes. Les classes :

- GraphicElements : qui contient en général des fonctions graphique outils
- GraphicAlgorithm : qui gère les différentes partie de l'algorithme du jeu en mode graphique en se servant des clics en guise d'entrée utilisateur
- GraphicPatch (interface scellée pour deux classes) : qui permet de représenter des patchs cliquables.
- GraphicButton : qui permet d'avoir des boutons cliquables.

La porte d'entrée du programme est la même. On utilise GameAlgorithme qui contient la boucle de jeu principale.

Dans le mode graphique nous avons tenté d'ajouter la fonctionnalité de parcourir le patchcircle et de visualiser le quiltboard du joueur actuel lors de la phase d'achat de patch.

Prise en comptes des remarques de la soutenance

Au niveau des fonctions « fourre tout », nous avons tenté de limiter ces méthodes et créer des sous méthodes pour déléguer certaines actions pour mettre à l'évidence le but principal de la méthode de départ.

Un exemple est la méthode : timeboardCrossingProcedure qui délègue la gestion individuelle de chaque élément spécial rencontré sur le timeboard à une autre méthode.

Au niveau des méthodes faisant plus de 20 lignes, il semblerait que nous avons corrigé ce problème avec la même solution que celle énoncée ci-dessus.

Pour essayer de moduler plus le jeu, nous avons également implémenter un système permettant de lire un fichier et puis construire un patchcircle.

En ce qui concerne le TimeBoard, on peut ajouter de la longueur en modifiant simplement les champs static de la classe. La version terminal peut très bien accepter ces modifications.

Cependant la version graphique est trop rigide à ce changement ce qui nécessite d'écrire une nouvelle méthode permettant d'afficher le timeboard. En effet nous avons eu du mal à essayer d'écrire une méthode permettant de dessiner un timeboard dynamiquement à partir de seulement la taille car cela

semble nécessiter beaucoup de temps et d'appréhension sur le problème pour obtenir une solution.

Philosophie de notre mode graphique

Nous tenons à préciser que nous avons eu du mal à démarrer sur la bibliothèque zen5. La manière dont les affichage étaient gérés nous a perturbé.

Les classes GraphicPatchStandard, GraphicPatchPlaced, GraphicPatchButton, GraphicAlgorithm ont un champs permettant de garder l'ApplicationContext utilisé. Ce qui nous permet de limiter les méthodes de ces classes à demander un ApplicationContext.

La porte d'entrée du programme reste la même, on commence par lire la classe Main qui recevra une instruction pour partir dans l'un des différents mode possible. Ce qui nous redirigera soit vers la méthode terminalMode ou graphicMode. Le mode terminal a très eu changé, nous nous concentrerons sur le mode graphique.

La boucle principale de jeu qui relance un tour à chaque itération est quasiment la même avec le mode graphique, en effet nous avons choisi de transposer le comportement du mode terminal vers le mode graphique en tentant de remplacer toutes les entrées sortie au terminal par un équivalent graphique. L'ApplicationContext pour l'affichage sera donc naturellement propagé entre appels de méthodes nécessitant de l'IO.

Pour mener à bien cette tâche nous avons alors au nécessairement besoin de boutons que l'on peut placer n'importe où à l'écran que l'on essayera d'associer à des instructions dans les méthodes. Ce détail est la motivation derrière la classe GraphicButton qui permet alors pour une instance d'afficher le bouton et de vérifier si les coordonnées d'un clic correspond à la surface où le bouton est dessiné.

Les commandes se font majoritairement autour des boutons hormis les phases de pose des patch et de sélections qui nécessitent de cliquer sur les patch pour les choisir ou bien de cliquer sur le quiltboard pour tenter de poser le patch.

Pour l'affichage des patchs, de la même manière que les boutons nous avons utilisé des Rectangle2D pour détecter un clic sur le patch. Et puis nous avons également implémenter des méthodes d'affichage du patch.

Cela nécessite d'encapsuler un patch vers un GraphicPatchStandard ou GraphicPatchPlaced.

La différence entre les deux sont :

- « Standard » est utilisé pour l'affichage du patchcircle alors que « Placed » sera utilisé pour afficher le quiltboard.
- Les blocs des « Standard » sont de tailles variables qui varient selon la taille totale du patch à encapsuler, tandis que les blocs des « Placed » sont tous de taille égale pour un ApplicationContext donné afin d'afficher de manière uniforme sur le quiltboard.
- La localisation du patch à afficher entre les deux sont différents, « Standard » utilise deux doubles pour localiser la position vis à vis de la fenêtre alors que « Placed » utilise un java.awt.Point pour localiser la patch sur le quiltboard.

Étant donné que les deux classes sont quand même très similaire en terme de comportement, nous avons alors décidé de créer une interface scellée afin de regrouper le code réutilisable en commun. (Qui sont deux méthodes permettant d'afficher les carrés colorés des blocs et d'afficher les contours des blocs).

L'affichage des messages se fait à travers de la méthode drawMessage de GraphicElements, d'autres méthodes qui affichent un message sont des dérivées de celle-ci pour des usages spécifiques (Affichage sur la sous fenêtre du bas pour les notifications de jeu, information sur le patch etc).

Recul sur la qualité de notre travail et architecture

En étant honnête, nous estimons que nous aurions pu faire mieux que cela. De manière plus précise, la partie graphique est malheureusement assez rigide contrairement au mode terminal où nous estimons que faire des modifications dessus serait plus simple.