Université Lumière Lyon 2

Rapport du projet de Poker planning

Renaud MACHECOURT-BOURGEOIS

Antoine ORUEZABALA

1. Justification des patrons de conception

Factory

Le patron de conception factory permet de définir une interface pour créer des objets dans une classe mère, dans le cadre de notre projet nous nous en sommes servis afin d'initialiser les images des cartes du poker planning rapidement et de manière automatisée.

Prototype

Le patron de conception prototype crée de nouveaux objets à partir d'objets existants sans rendre le code dépendant de leur classe, nous en avons fait usage pour créer des objets des cartes à partir d'un prototype, ce qui nous permet d'avoir un meilleur temps de traitement.

State

Le patron de conception state est un patron de conception comportemental qui permet de modifier le comportement d'un objet lorsque son état interne change. L'objet donne l'impression qu'il change de classe. Pour notre projet, il nous permet de suivre à quel état de la partie nous somme, on pourra donc l'enregistrer dans nos fichier JSON simplement.

2. Justification des choix techniques

Le langage de programmation

Pour ce qui est du langage de programmation, nous avons décidé de partir sur le langage C. C'est un langage sur lequel nous étions plus à l'aise pour coder.

L'interface

Pour ce qui est de l'interface graphique, nous avons fait le choix d'utiliser la bibliothèque SDL (Simple DirectMedia Layer), qui est très utile dans le langage de programmation que nous avons choisi. Nous nous étions déjà servis de SDL par le passé, il nous a donc paru logique de le réutiliser pour ce projet.

Les fichiers JSON

En ce qui concerne le format du JSON, nous avons décidé de partir sur cela : Mode de jeu ; État ; Nombre de tâches restant à évaluer ; Dernière tâche évaluée ; Tâche n°1, [Note] (si décidé) ;

Les informations que nous avons décidé de sauvegarder dans les fichiers JSON sont celles qui nous ont parues les plus pertinentes afin d'avoir une idée précise d'où en est la partie au moment de la sauvegarde.

Nous avons décidé de ne pas sauvegarder le nom des participants car ces derniers pourraient changer. De plus, l'ordre des tâches n'a pas d'importance à l'exception qu'il faut que toutes les tâches déjà évaluées soient en premières et les autres après. On enregistre le nombre de tâche total ainsi que le nombre de tâches déjà faites afin de ne pas avoir tout le fichier à parcourir en redémarrant la partie, ce qui nous permet de gagner du temps de calcul.

Décision des tâches à accomplir

Afin de mener à bien notre projet, nous avons pris la décision de lister les différentes tâches qu'il faudra faire au cours de notre projet et de les classer en effectuant un Poker Planning de notre projet.

De la plus simple à la plus compliquée, les tâches que nous allons faire dans notre projet sont :

- Le menu pour choisir le mode de jeu en sachant que le mode Strict sera appliqué par défaut lors du 1er tour, avec le mode moyenne, médiane, majorité absolue et majorité relative.

- Une fonction pour permettre aux joueurs de choisir leurs pseudos

- Un chronomètre pour délimiter la longueur d'une partie

- L'enregistrement de la partie ainsi que le démarrage d'une partie via un fichier JSON

- La mise en place d'un chat pour permettre aux joueurs de noter des informations. Notre application n'étant utilisable qu'en local, les joueurs peuvent se parler directement.

- Mettre en place le système de vote

Structure du code

Nous avons décidé de structurer notre code en le scindant en différents fichiers spécifiques, il y a les fichiers dans lesquels on retrouve le code des patrons de conception, le fichier de l'environnement le fichier du jeu en lui-même… De cette manière on peut vérifier que chaque fichier fonctionne individuellement et suite à cela. Si un fichier à besoin du code d'un autre, on peut l'appeler via la commande #include.

3. Intégration Continue

Génération de documentation

La documentation du code peut paraître ennuyeuse, mais elle est importante pour que les personnes qui consulte notre code ne soient pas perdues, il faut donc qu'elle soit claire et utile. Nous avons commenté de nous-même les parties les plus importantes de notre code et pour le reste nous avons décidé de documenter de manière automatisée avec un outil de génération de documentation Doxygen. Nous avons choisi cet outil car il était compatible avec notre langage de programmation et nous savions comment l'intégrer à notre code.

Tests Unitaires

Les tests unitaire permettent de vérifier le bon fonctionnement d'une partie précise d'un logiciel ou d'une portion d'un programme. Dans le cadre de notre projet nous n'avons pas eu le temps de faire l'intégration continue nous permettant de générer les tests unitaires.