Université de Technologie de Compiègne
LO21, A21

Rapport Final
Projet LO21 : Splendor

Solène Desvaux de Marigny - Pierre Fagot
Thomas Lerner - Gabrielle Van de Vijver

Janvier 2022

Desvaux-Fagot
Lerner-Van de Vijver

Sommaire

Desvaux-Fagot
Lerner-Van de Vijver

I - Introduction

Dans le cadre de l'UV LO21, il nous est demandé de concevoir et développer une application en C++. Ce semestre, l'objectif de l'application est de permettre à une ou plusieurs personnes de jouer au jeu de société *Splendor,* créé par Marc André et édité par *SPACE Cowboys*.

Ce projet a été réalisé par une équipe de quatre étudiants de GI de l'UTC.

Pour ce projet, nous avons utilisé un répertoire commun sur le Gitlab UTC. Nous avons ainsi pu nous familiariser avec l'outil et exploiter les fonctionnalités d'un outil de versionnage.

Desvaux-Fagot
Lerner-Van de Vijver

II - Résumé de l'application

Notre application Splendor a donc le principe suivant :

Une interface graphique, réalisée en utilisant de framework *Qt*, permet au(x) joueur(s) d'interagir avec le code.

En lançant l'application, une fenêtre de paramétrage de la partie apparaît. Cette dernière permet d'indiquer le nombre de joueurs pour la partie (de 2 à 4), ainsi que de préciser si ce sont des IAs ou des humains.
Il est ensuite possible d'ajouter une extension à la partie. Nous avons décidé d'implémenter l'extension *Cities*.

Une fois la partie lancée, chaque joueur joue chacun son tour. Sur l'interface sont affichées les cartes faces visibles, les nobles, les pioches et les piles de jetons. A chaque tour, un joueur peut décider de récupérer des jetons, acheter ou réserver une carte, selon les règles du jeu d'origine.

A chaque tour, les points de prestige des joueurs sont calculés. La partie s'arrête lorsqu'un joueur à atteint 15 points de prestige.

A ce moment, le tour se termine et on affiche le gagnant.

Dans la pratique, l'implémentation n'a pas été complète. Dans l'ensemble, le fonctionnement de l'application (fonctionnement du plateau, gestion des joueurs et des objets associés) a été fait. Les différentes fenêtres sont créées et ont bien été implémentées. L'affichage se fait correctement et les objets sont initialisés comme prévu.

Nous n'avons malheureusement pas réussi à rendre le jeu fonctionnel. Il est probable que ce soit par faute de temps consacré à la mise en relation back/front. Cela peut s'expliquer par le fait que nous avons voulu faire une interface agréable à utiliser, ce qui a détourné notre attention de la fonctionnalité basique.

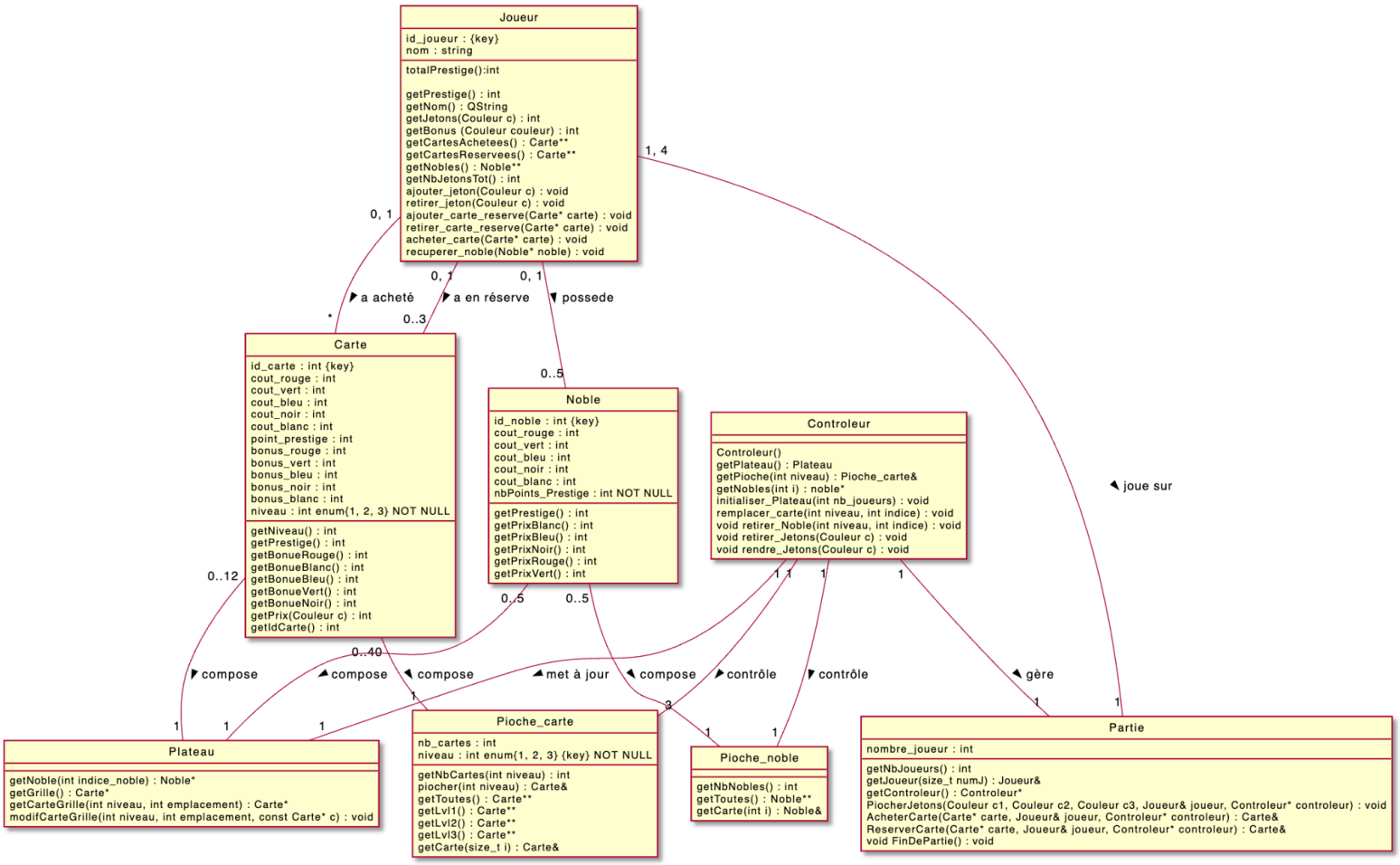
De plus, de part le retard que nous avons pris, nous avons préféré nous concentrer en priorité pour faire fonctionner une partie classique (sans IA ni extension). Nous n'avons donc pas pu nous pencher assez sur ces parties de code pour obtenir un bot ou une extension fonctionnelle.

Desvaux-Fagot
Lerner-Van de Vijver

III - Description de l'architecture

Nous avons décidé de séparer, afin de mieux nous répartir le travail, l'implémentation de l'interface utilisateur (UI), et le développement des fonctionnalités du programme.

L'ensemble des classes du programme sont les suivant on été représenté dans l'UML suivant :



Cette version de l'UML est la dernière que nous avons produite et est celle qui représente le plus fidèlement la structure de notre projet.

Chacune des classes est décrite dans un document *doc.h* que vous retrouverez dans notre code source, et un document *doc.cpp* lui est associé.

a. Plateau

Une fois la partie lancée, et donc le nombre de joueurs validés, le plateau est créé. La classe *Plateau* comprend une grille de 12 cartes au début du jeu (4 par niveau), ainsi que les nobles (le nombre de nobles varie en fonction du nombre de joueurs).

b. Partie

Notre code s'articule autour de la classe *Partie*, qui gère les tours de chaque joueur.
Une fois le jeu lancé et le plateau créé, Partie donne la main aux joueurs un à un. Le joueur choisit entre les 3 actions possibles (piocher des jetons, acheter ou réserver une carte).
C'est la *Partie* qui vérifie à chaque fois si un joueur coche tous les critères pour être visité par un noble, et termine la partie lorsqu' un joueur a plus de 15 points de prestige.

c. Contrôleur

Le contrôleur est un élément central de notre programme. Il est chargé de mettre à jour le plateau au fur et à mesure de la partie. Il regroupe le plateau, mais aussi les pioches et les piles de jetons. Il garde en mémoire le nombre de jetons dans chaque pile et est chargé de sortir une nouvelle carte de la pioche pour l'afficher sur le plateau à chaque fois qu'une carte est achetée ou réservée.

d. Joueur

La classe joueur comprend toutes les informations qui correspondent au jeu d'un joueur : son nombre de jetons par couleur, ses cartes achetées ou réservées, ainsi que ses points de prestige.

e. Carte

Une carte est composée d'un id, d'un niveau, d'un nombre de points de prestige attribués au joueur qui l'achète, d'un prix par couleur (ex : 5 jetons bleus, 3 rouges) et d'un bonus (aussi appelé remise) qui correspond au jeton que la carte nous rapporte.

f. Noble

Un noble est composé d'un nombre de points de prestige qu'il rapporte au joueur qui récupère la carte, et d'un coût nécessaire pour qu'il vienne visiter un joueur.

g. Pioche_carte

Cette classe regroupe les 3 pioche (une par niveau de carte) qui pointent sur les cartes d'un même niveau de manière aléatoire, pour les mélanger

h. Pioche_noble

De la même manière, cette pioche pointe sur les cartes nobles de manière aléatoire pour les afficher sur le plateau en début de partie.

i. Exceptions

La classe exception, définie dans *exception.h*, renvoie un message qui justifie l'impossibilité d'effectuer une action.
Nous avons implémenté deux types d'exceptions

- Exceptions explicites, directement liées aux règles du jeux

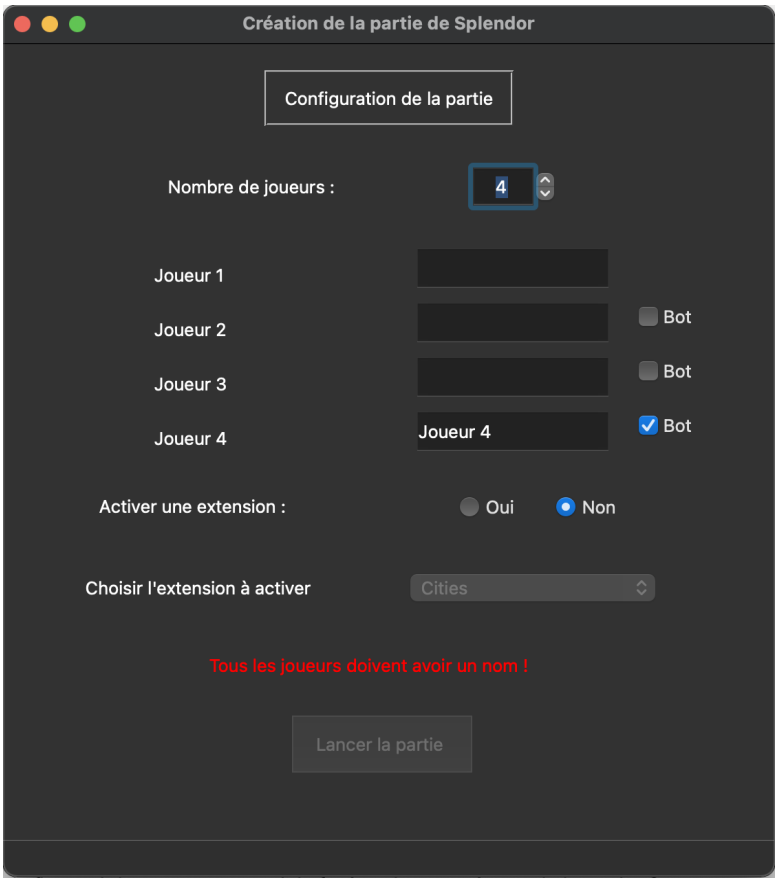
Ex : interdiction de prendre deux jetons de même couleur s'il en reste moins de quatre disponibles dans la banque centrale de jetons

- Exceptions implicites, liées aux limites matérielles du jeu

Ex : impossible de piocher une carte si la pioche en question est vide

j. Interface graphique

Desvaux-Fagot
Lerner-Van de Vijver

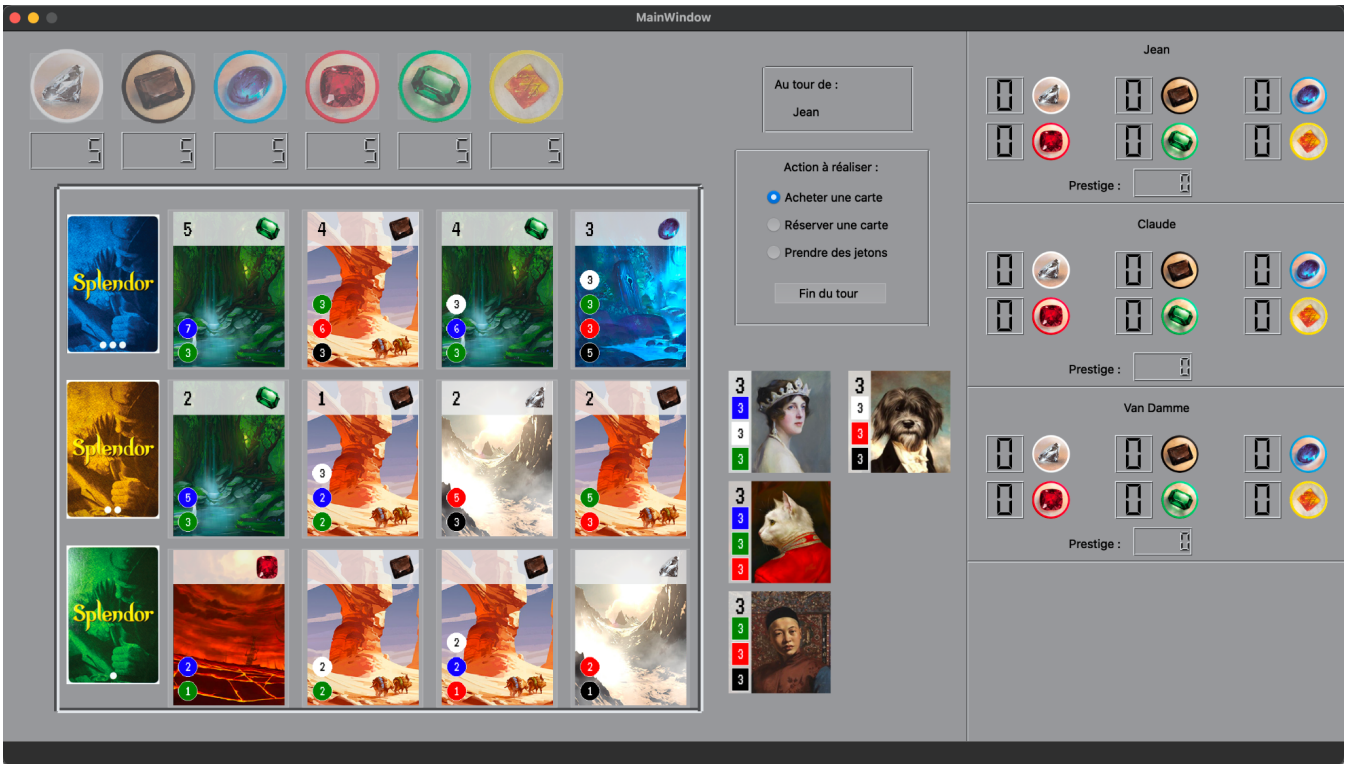


Sur la figure ci-dessus, on peut voir la fenêtre de paramétrage de la partie. On y retrouve une QSpinBox permettant de sélectionner le nombre de joueurs. En dessous, un input permet d'inscrire le nom de chaque joueur, ainsi que de choisir si le joueur doit être géré par une IA ou non. Dans le cas où l'on choisirait une IA, le nom du joueur est automatiquement rempli. Un radio bouton permet aussi de choisir si l'on souhaite jouer avec une extension et une liste en dessous permettant de choisir l'extension voulue.

Un avertissement s'affiche dans le cas où l'on essaierait de lancer une partie avec des joueurs sans nom. Le bouton "Lancer la partie" permet alors d'avancer et de lancer la partie.

On clique alors sur le bouton "Lancer la partie" et cela lance la fenêtre de jeu, visible ci-dessous :

Desvaux-Fagot
Lerner-Van de Vijver



L'interface se découpe en plusieurs éléments :

- Le plateau, composé des 3 rangées de cartes et de leur pile de cartes de niveau respectif. C'est le centre du jeu, c'est ici que tout s'articule.
- Les piles de jetons, au-dessus du plateau, dans lesquelles les joueurs peuvent prendre des jetons pour pouvoir après acheter des cartes.
- Une petite case signalant le joueur dont c'est le tour.
- En dessous de cette case, une autre permet de sélectionner une action de jeu entre acheter une carte, réserver une carte ou prendre des jetons. Choisir une des actions rend les autres impossibles. Le bouton Fin du tour permet de valider la sélection et de passer la main au joueur suivant.
- Sous cette case s'affichent les nobles de la partie. Il n'y a pas d'action de jeu relative à ces cartes, elles ne sont donc que des images et le joueur ne peut pas interagir avec.
- Tout à droite, on retrouve la colonne des joueurs. Cette colonne n'affiche que le nombre de joueurs défini en paramétrage de la partie. On y retrouve leur nom, leur nombre de jetons de chaque couleur et leur prestige actuel.

Desvaux-Fagot
Lerner-Van de Vijver

IV - Gestion des évolutions

a. Classes

Pour chaque document *doc.h*, les attributs de nos classes sont privés dans les classes et ne sont accessibles que par des méthodes, afin de protéger leur manipulation. Nous avons essayé d'implémenter chaque classe de la façon la plus indépendante possible des autres pour permettre de modifier, ou remplacer les fonctions le plus simplement possible, sans avoir à aller modifier tous les autres fichiers du projet, avec en tête la perspective d'implémenter de nouvelles fonctionnalités ou de nouvelles extensions.

b. Singleton

Afin de n'être lancé qu'une seule fois par partie, nous avons décidé d'implémenter un singleton de la classe *Partie*. Cela permet aussi d'avoir une instance récupérable depuis tout document du code. Ainsi, il est simple de récupérer des informations et les attributs de nos objets pour les exploiter, où que l'on soit dans le code.

c. Séparation Plateau/Controleur/Partie

Cette séparation permet tout d'abord de mieux contrôler l'influence du nombre de joueurs sur le nombre de jetons initiales et le nombre de nobles.

De plus, cela permet un meilleur contrôle de modification des règles, donc l'ajout d'extension par exemple.

d. Utilisation d'un document JSON

Le fait que les cartes soient générées depuis un document JSON permet de changer le deck de carte de manière plus facile. Le code lit le document JSON et génère toutes les cartes et les nobles de manière automatique. Il suffit pour cela de faire les modifications dans le documents JSON correspondant en faisant attention de respecter la structure de ce dernier.

En effet, l'activation de l'extension *Cities* par exemple nécessite de changer les cartes Noble et par des cartes Villes, ce qui est tout à fait faisable avec ce format. L'avantage de l'utilisation d'un fichier JSON est la versatilité de ce genre de format, de la facilité de modification et que si l'on souhaite remplacer notre jeu de cartes, c'est facilement faisable et ce, sans avoir à modifier le code.

De même, il est facile de faire évoluer les graphiques du jeu, il suffit de changer les images dans le fichier ready et le code s'occupe de les lires.

V - Planning détaillé

Nous avions commencé le projet en découpant les tâches et en les répartissant entre les membres du groupe. Cette répartition a bien entendu été mise à jour au fur et à mesure de l'avancée du projet.

Notre planning de tâches a été le suivant :

Nom de la tâche	Durée	Personne en charge	Deadline	Commentaires
Général				
Apprendre les règles du jeu	~3h	Tous	15/11	En collaboration avec le CoinDuJoueur
Etudier les design pattern	~3h	Gabrielle	11/12	
Ouvrir un git	~1 min	Thomas	01/12	
UML	~2h	Tous	08/12	Mis à jour régulièrement
Faire le rapport 1	~1h	Tous	15/11	
Faire le rapport 2	~1h	Tous	06/12	
Faire le rapport 3	~1h	Gabrielle	20/12	
Coder les fichier .h				
Coder Carte.h	~1h	Pierre	1/12	
Coder Joueur.h	~1h	Gabrielle	1/12	
Coder Noble .h	~1h	Solène	1/12	
Coder Plateau.h	~1h	Pierre	1/12	
Coder Controleur.h	~1h	Gabrielle	14/12	Ajouté après le rapport 1
Coder Pioche_Carte.h	~1h	Solène	14/12	Ajouté après le rapport 1
Coder Pioche_Noble.h	~1h	Pierre	14/12	Ajouté après le rapport 1
Coder Partie.h	~1h	Pierre	1/12	
Coder Jeton.h	~1h	Thomas	30/11	Annulé
Coder Pile_jetons.h	~1h	Thomas	1/12	Annulé
Coder Exception.h	~1h	Gabrielle	14/12	Ajouté après le rapport 1
Coder Gamewindow.h	~1h	Thomas	14/12	Ajouté après le rapport 1
Coder Global.h	~1h	Pierre	14/12	Ajouté après le rapport 1
Coder mainwindow.h	~1h	Thomas	14/12	Ajouté après le rapport 1

Coder les fichier .cpp				
Coder Carte .cpp	~2h	Pierre	1/12	
Coder Joueur.cpp	~2h	Gabrielle	1/12	
Coder Noble .cpp	~2h	Solène	1/12	
Coder Plateau.cpp	~2h	Pierre	1/12	
Coder Controleur.cpp	~2h	Gabrielle	21/12	Ajouté après le rapport 1
Coder Pioche_Carte.cpp	~2h	Solène	21/12	Ajouté après le rapport 1
Coder Pioche_Noble.cpp	~2h	Solène	21/12	Ajouté après le rapport 1
Coder Partie.cpp	~2h	Pierre	1/12	
Coder Main.cpp	~30min	Solène	1/12	
Coder Jeton.cpp	~1h	Thomas	1/12	Annulé après changement de modèle
Coder Pile_jetons.cpp	~1h	Thomas	1/12	Annulé après changement de modèle
Coder gamewindow.cpp	~1h	Tous	21/12	Ajouté après le rapport 1
Mise à jour des ressources				
Carte.json	~1h	Pierre	21/12	
Noble.json	~1h	Solène	21/12	
Interface graphique				
mainwindow.ui	~5h	Thomas	24/12	
gamewindow.ui	~5h	Thomas	24/12	
Mise en commun				
Vérification des codes	~5h	Solène/Pierre	28/12	
Uniformisation des codes	~5h	Pierre/Gabrielle/ Solène	31/12	
Débogage	~15h	Pierre/Gabrielle/ Solène	31/12	
Liaison UI/backend	~10h	Thomas/Pierre	31/12	
Etapes supplémentaires				
Extension	~5h	Tous		
IA	~10h	Tous		
Sauvegarde du contexte	~5h	A définir	2/1	Annulé

Historique des scores	~5h	A définir	2/1	Annulé
Livrables finaux				
Rédiger le rapport	~5h	Solène	3/1	
Enregistrer la vidéo	~1h	Tous	3/1	
Documentation Doxygen	~8h	Solène	3/1	
Compléter le rapport	~2h	Tous	3/1	
Vérification doxygen/code source	~1h	Pierre	3/1	

Comme vous pouvez le constater, un des défauts majeurs de cette organisation a été la mise en commun des morceaux de codes pendant les vacances. De par la distance, et les incompatibilités d'emplois du temps, nous avons pris du retard sur l'implémentation générale du code. En effet, chaque partie était prête indépendamment, mais nous avons sous estimé le temps nécessaire à l'uniformisation des variables, et à la liaison de l'interface graphique au back end.

Nous pensions réaliser les étapes qui en découlent par la suite (développement de l'IA, de l'extension etc…) mais en raison du retard que nous avons pris, nous avons fait le choix de nous concentrer en priorité sur le fonctionnement global d'une partie classique.

VI - Contribution personnelle des membres du groupe

a. Solène Desvaux de Marigny

Lors du début du projet, j'ai commencé par coder les fichiers *Noble* et Pioche_Carte (.h et .cpp), ainsi que la mise à jour du dossier JSON correspondant aux ressources nobles.

La partie que nous avons le plus sous-estimée était la mise en commun des fichiers. J'ai donc ensuite travaillé en grande partie sur la vérification et l'uniformisation des codes.

Quant aux différents rendus, j'ai participé à la rédaction des rapport 1 et 2, et j'ai travaillé sur le rapport final, ainsi que la documentation du code en *Doxygen*.

J'estime avoir passé une tout une soixantaine d'heures sur le projet.

b. Pierre Fagot

J'ai commencé par travailler sur Carte (.h et .cpp) j'ai ensuite travaillé sur le plateau et la pioche noble.
J'ai ensuite passé beaucoup de temps à vérifier les codes, ajouter les fonctionnalités manquantes et uniformiser le code. J'ai pu ensuite m'occuper de Partie (la classe principale qui fait le lien entre le joueur et le contrôleur.
Ensuite, nous avons commencé à faire le lien entre l'interface et le back-end. Nous avons largement sous-estimé cette partie ce qui nous fait prendre beaucoup de retard.
Finalement, j'ai participé à la rédaction de la documentation en doxygen.

Quant aux différents rendus, j'ai participé à la rédaction des deux premiers rapport intermédiaires et du rapport final.

J'estime avoir passé entre 80 et 90 heures sur le projet.

c. Thomas Lerner

Au début du projet, j'ai commencé par implémenter les jetons. En avançant sur le projet, nous avons décidé que ce modèle ne faisait qu'un objet supplémentaire à gérer. Nous avons donc supprimé cet élément de notre modèle.

J'ai alors commencé à implémenter les fenêtres Qt. D'abord la fenêtre de paramétrage de la partie, qui aurait vocation à initialiser les joueurs et la partie avec les paramètres que nous souhaitons, puis la fenêtre de jeu, que nous avons voulue simple et la plus épurée possible. L'idée était de faire au plus simple et au plus visuel : la décision prise a donc été de rendre les cartes et les jetons cliquables en vue de rendre l'utilisation la plus simple possible et la plus intuitive. Il semblerait que cette idée était meilleure sur le papier que dans la pratique car la gestion des signaux et des variables était compliquée à implémenter.

En ce qui concerne les rendus, j'ai participé à la rédaction des rapports intermédiaires et du rapport final également.

d. Gabrielle Van de Vijver

Au début du projet, j'ai travaillé sur la modélisation conceptuelle du jeu (UML). J'ai étudié les ressources à notre disposition (design pattern, codage du set) pour que l'on puisse s'en inspirer pour l'implémentation. J'ai codé les fichiers joueur.h, joueur.cpp et controleur.cpp. J'ai travaillé sur l'harmonisation des codes suite à la finalisation de mes fichiers controleur, puis j'ai participé à la correction du fichier partie.cpp. Pour ce qui est des rapports, j'ai participé à la rédaction des rapports 1,2, du rapport final et je me suis chargée du rapport intermédiaire 3. Temps de travail estimé : > 55h.

e. Part de contribution de chaque membre

- Solène : 20%
- Pierre : 40%
- Gabrielle : 20%
- Thomas : 20%

Desvaux-Fagot
Lerner-Van de Vijver

VII - Extension & bot

Sur la fin du projet, nous avons choisi de prioriser le débogage du code pour opérationnaliser la version classique du jeu Splendor, soit sans extension activée et avec des joueurs humains.
Voici comment nous aurions pu aller plus loin :

Notes pour développer l'extension Cités :
Niveau conceptuel :
Classe abstraite tuile avec noble et cité en classe fille
Stratégie d'implémentation:
En début de partie, possibilité d'activer ou non l'extension, ce qui est retransmis au constructeur de contrôleur à l'aide d'un booléen. Les méthodes de gestion des tuiles cités sont semblables à celles déjà définies pour nobles, à la différence que le plateau est initialisé avec 3 tuiles cités (le nombre de joueurs n'est pas un paramètre) et que les combinaisons de cartes + les points de prestiges sont pris en compte pour déterminer si un joueur est éligible à une cité.
Dans partie, à la fin de chaque tour de joueurs, on regarde combien d'entre eux ont reçu une tuile cité

si o -> poursuite de la partie
si 1 -> fin de la partie, le joueur en question est vainqueur
si >1-> comptage des prestiges par joueur pour les départager

Notes pour développer un bot :
Niveau conceptuel :
Création de classes humain et bot qui héritent de joueur.
Stratégie d'implémentation
On définit, pour chacune des actions du jeu, le comportement du bot (méthodes spécifiques dans partie). On peut dans un premier temps concevoir des bots qui choisissent des actions de façon aléatoire jusqu'à aboutir à une action qui ne déclenche pas d'exception.

Pour associer au bot des comportements intelligents, il faudrait déjà implémenter une méthode, déclenchée avant chaque action des bots, qui étudie la situation du bot et la configuration de la partie de façon à ne retenir que les actions possibles pour le bot (sans déclenchement d'exception). Le bot sélectionne alors, au hasard, une action parmi celles-ci.

A terme, il serait pertinent de réduire la part de hasard dans les actions du bot, et de redéfinir son comportement de façon à reproduire le comportement d'un joueur expérimenté. Nous pourrions par exemple faire en sorte que le bot vise, à chaque tour, une carte du plateau ou de sa réserve, et qu'il pioche ses jetons en conséquence. (mais nous sommes limités sur cette question car nous sommes nous-mêmes des joueurs débutants de Splendor). Il serait intéressant enfin de développer un système de machine learning, qui permettrait au bot d'améliorer sa stratégie de jeu au fur et à mesure des parties jouées.

Nous pourrions aussi essayer d'utiliser un bot basé sur l'algorithme MinMax, qui pourrait alors évaluer les scénarios à perte minimale afin d'optimiser au maximum ses actions de jeu et s'assurer un avantage contre un joueur humain.

VIII - Conclusion

Ce projet nous a permis d'apprendre à travailler en équipe, et nous a surtout permis de nous rendre compte de ce que représente un projet de développement d'application en groupe. Il y a de nombreux facteurs à prendre en compte. Suivre un même projet à plusieurs avec des emplois du temps et des compétences différentes n'est pas évident, mais nous avons fait de notre mieux pour outrepasser les obstacles induits par nos différentes façons de travailler.

Nous avons maintenant compris l'importance du travail en amont, sur la préparation et l'organisation du code, afin de gagner du temps sur la mise en commun et éviter des erreurs de morceaux de codes "doublons". Nous avons aussi compris l'importance capitale de s'accorder sur des nomenclatures communes afin de ne pas avoir à passer de nombreuses heures à essayer de tout harmoniser et normaliser.

Enfin, et cela était l'objectif principal, nous avons profité de ce projet pour nous familiariser avec la programmation orientée objet, le C++, ainsi que l'outil de documentation *Doxygen* et le framework *Qt*.