Hugo PEREIRA, Clément MARTINS, Cyril PETER, Hans HOOKOOM, Alexandre BIDAUX

AI22/LO21 – Splendor Duel

Rapport Final

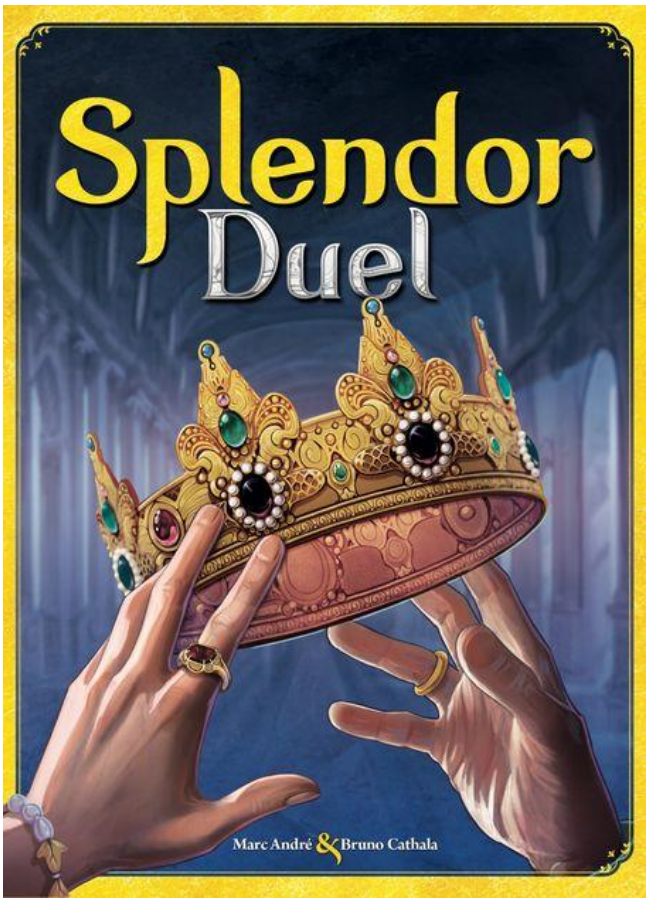


TABLE DES MATIERES

RESUME DE CE QUE PERMET NOTRE APPLICATION

Depuis début septembre, nous travaillons sur une application graphique permettant de jouer au jeu Splendor Duel sur une machine.

Cette application est aujourd'hui terminée et permet les actions suivantes :

- Jouer contre un joueur adverse
- Jouer contre une IA rudimentaire
- Changer la langue du jeu

Une fonctionnalité que nous n'avons pas eu le temps de finir à cause d'une trop grosse dette technique est la possibilité de sauvegarder sa partie. Les composants de sauvegarde existent et fonctionnent mais afin de gagner du temps dans la conception nous avons fait un découplage trop fort entre l'UI et les composants logiques du jeu ce qui empêche une modification en direct des composants logiques impactant l'UI.

Un choix que nous avons fait dans le code est de représenter les réductions comme étant une gemme en plus dans l'inventaire du joueur. Donc avoir une réduction de 1 sur les gemmes bleues par exemple se traduit par un ajout de 1 constant au stock courant de gemmes bleues (si 3 gemmes bleues en possession, 4 sera affiché).

Pour jouer au jeu :

- Sélectionner un jeton avec click gauche
- Acheter une carte avec click gauche
- Réserver une carte avec click droit
- Utiliser un privilège avec le click (droit ou gauche)
- Scroller ou utiliser la touche L pour ajuster la longueur de votre sélection de gemmes
- Click droit ou utiliser la touche N pour changer l'orientation de votre sélection de gemmes

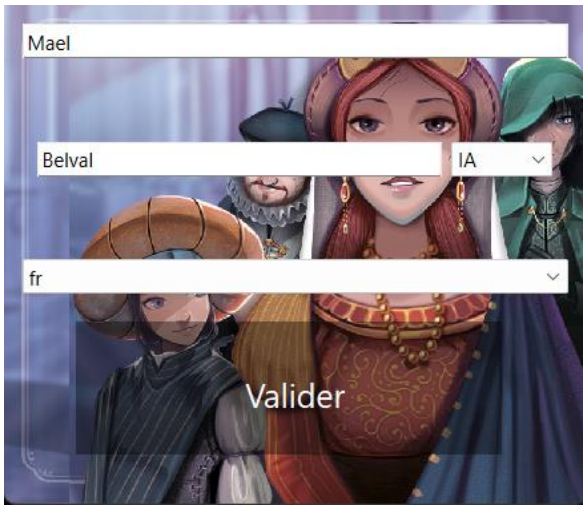


Figure 1 : Capture d'écran du menu de lancement

Figure 2 : Capture d'écran d'une partie en cours

DESCRIPTION DE NOTRE ARCHITECTURE

DESCRIPTION GENERALE

Notre architecture est décomposée en 3 grandes briques :

- Composants logiques
- Composants UI
- Classes utilitaires

Le cœur de l'application repose sur les composants logiques (et certains utilitaires desquels dépendent certaines classes logiques) qui ont les capacités de gérer leurs interactions avec les autres classes logiques ainsi que leurs propres attributs.

En théorie, nos composants UI viennent agréger nos composants logiques et servent d'intermédiaires pour l'affichage et l'interaction avec ces classes. Cependant, nous avons accumulé une dette technique sur nos composants UI en ne les faisant que s'appuyer légèrement sur les classes logiques. Par exemple, la classe servant à afficher les cartes possédées par un joueur ne se met pas à jour à travers les cartes possédées par la classe Player mais à travers le click de l'utilisateur. Il est tout à fait logique que lors de l'achat d'une carte, nous agrégions la carte sélectionnée à notre classe logique, ce qui se répercutera dans la classe UI. Ici, nous ajoutons la carte à notre classe logique et directement à notre classe UI qui ne dépend donc pas de la classe UI pour connaitre les cartes à afficher. Cela nous a permis de plus rapidement développer notre application mais a mené à de la redondance d'information et surtout un découplage entre les composants logiques et UI qui nous ont bloqué pour l'ajout de certaines fonctionnalités.

Voici un MCD représentant notre architecture actuelle :

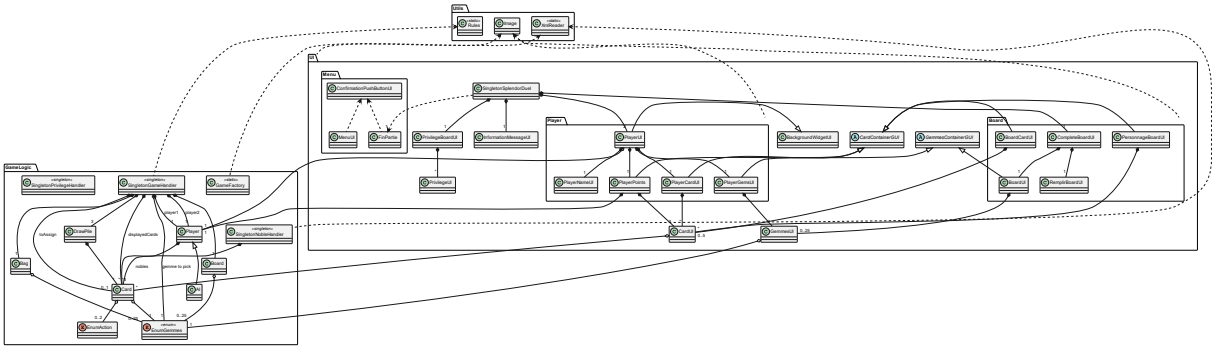


Figure 3 : MCD de notre architecture

COMPOSANTS LOGIQUES

Nos composants logiques sont chargés au lancement de l'application avec le nombre par défaut de chaque élément du jeu.

A l'aide d'un fichier XML nommé cards.xml, on génère l'intégralité des cartes du jeu. Nous répartissons ensuite ces cartes dans les différentes piles de cartes. Les piles de cartes sont ensuite mélangées et distribuées sur le plateau de jeu.

On génère également les 25 gemmes du jeu dans les proportions de base (4 gemmes bleu, blanc, rouge, vert, noir ; 3 gemmes or ; 2 gemmes perles) et on les met dans le sac du jeu. On mélange le sac pour ensuite remplir le plateau du jeu avec les gemmes. Le plateau du jeu étant complexe à naviguer (en spirale à partir du centre), on a créé un itérateur qui permet de naviguer correctement le plateau tout en ne partageant pas la structure de données.

On crée ensuite les joueurs avec leurs noms et aucune possession. Si l'utilisateur a choisi d'affronter une IA, on remplace un des joueurs par une IA nommée « IA ».

A partir de ce moment, on instancie les classes SingletonSplendorDuel et SingletonGameHandler qui sont respectivement les classes principales pour l'UI et la logique du jeu.

Cette génération à lieu dans une Factory, le but est de pouvoir plus tard remplacer cette Factory par une autre dans le cas où le jeu serait amené à évoluer.

Voici un MCD présentant notre architecture logique :

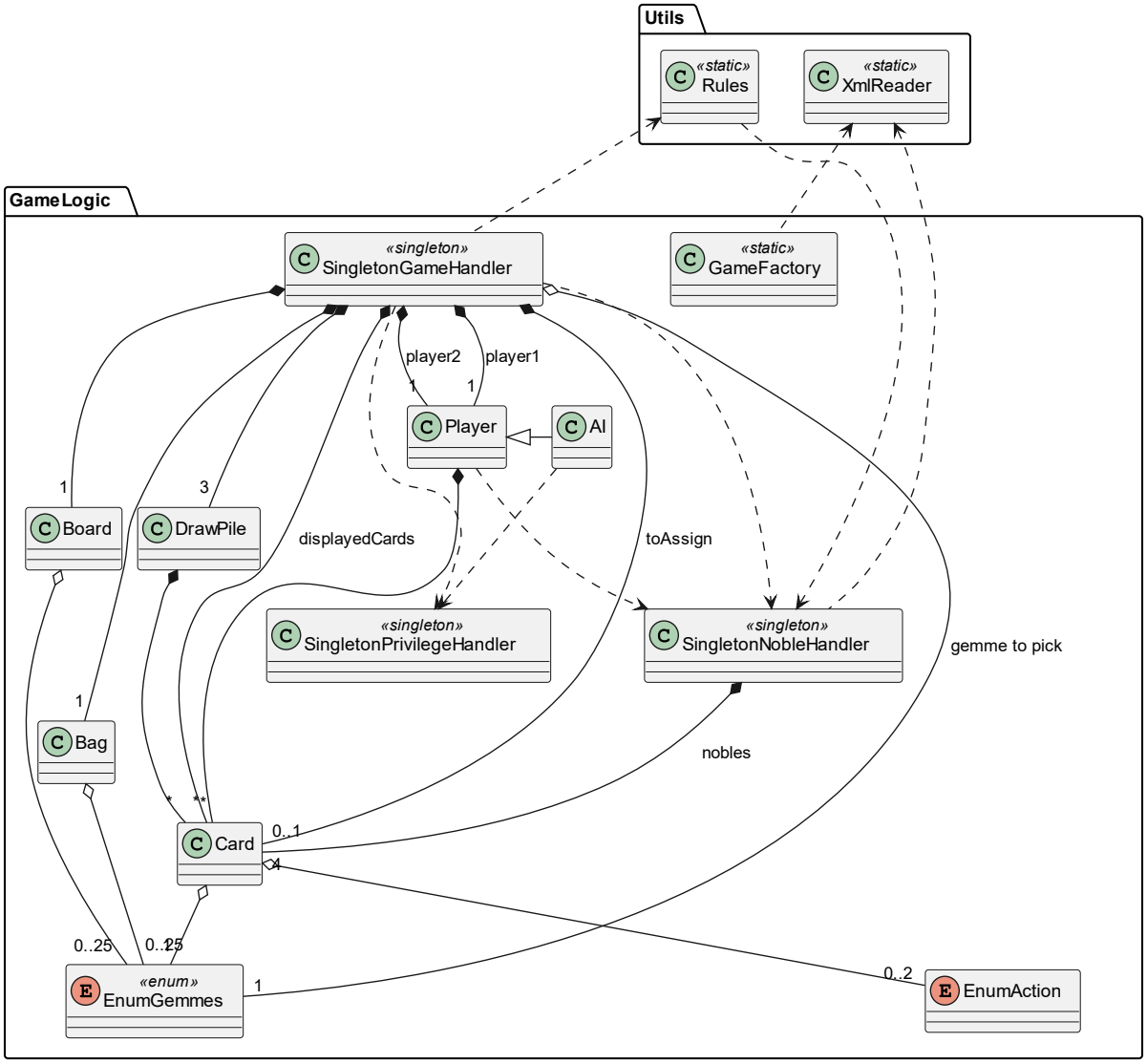


Figure 4 : MCD de notre architecture applicative

COMPOSANTS UI

Nos composants UI servent à gérer les interactions entre l'utilisateur et la logique.

Ainsi, nous répartissons nos composants UI en 4 parties :

- Les composants « Généraux »
- Les composants « Board » qui concernent le plateau
- Les composants « Menu » qui concernent les différents menus du jeu
- Les composants « Player » qui concernent le joueur

Les composants « Généraux » sont appelés un peu partout dans l'UI (on peut penser aux cartes par exemples qui sont affichées chez le joueur et sur le plateau de jeu). On y retrouve des classes abstraites comme CardContainerGUI et GemmesContainerGUI qui imposent la gestion d'évènements comme click ou le scroll sur d'autre composants UI. On y retrouve la classe SingletonSplendorDuel qui est la classe mère de toute l'UI du jeu.

Les composants « Board » concernent l'affichage du plateau, on y retrouve les classes UI chargées de gérer les cartes affichées sur le plateau (BoardCardUI), le plateau avec les gemmes (BoardUI) ou encore la gestion de l'affichage des cartes nobles (PersonnageBoardUI).

Les composants « Menu » concernent les petits menus rapides qui s'affichent au lancement de l'application et à la fin d'une partie.

Les composants « Player » concernent l'affichage des informations concernant les joueurs. On y retrouve des classes chargées d'afficher les cartes du joueur (PlayerCardUI), ses points (PlayerPointsUI) ou plus simplement son nom (PlayerNameUI). La classe qui compose toutes les classes de cette catégorie est PlayerUI.

Voici un MCD présentant notre architecture UI :

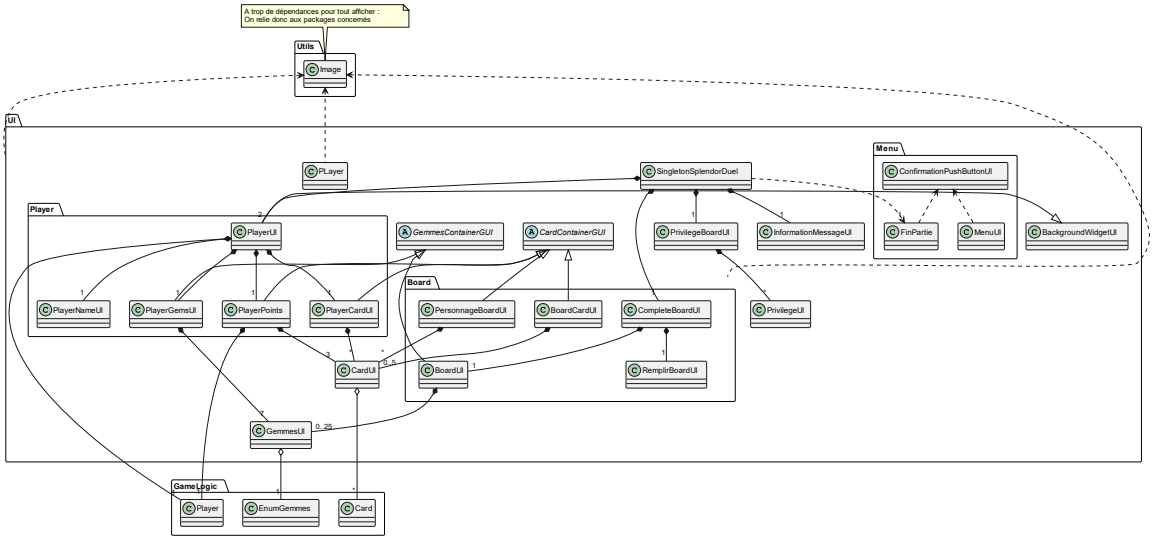


Figure 5 : MCD de notre architecture UI

MODULARITE DE L'ARCHITECTURE ET AMEILIORATIONS POSSIBLES

Notre architecture est assez permissive en ce qui concerne la modification des classes. Cela est principalement dû à la GameFactory qui a comme responsabilité l'instantiation des classes. Si par exemple je modifie la classe Card (qui a de nombreuses dépendances) de telle sorte a ce que son interface reste la même mais que ses attributs ou sa construction change, nous n'avons qu'à modifier la GameFactory pour que la modification soit faite correctement. De plus, nous pouvons ajouter/retirer des cartes simplement en modifiant le fichier cards.xml qui les gère.

L'aspect le moins modulaire de la partie logique réside dans la taille du sac de gemmes. Celui-ci est hard coded comme étant limité à 25. Ce qui veut dire que l'ajout de plus de jetons dans la partie demandera plus de travail qu'un rapide remaniement de la classe GameFactory.

Cependant, comme déjà mentionné dans ce rapport, nous avons accumulé de la dette technique sur la partie UI qui est trop découplé de la partie logique. La partie UI devrait dépendre beaucoup plus de la partie logique et se reposer sur les fonctions de calcul des classes logiques plutôt que de recalculer les différentes opérations de l'utilisateur.

Cependant, nous pouvons également ajouter facilement des langues pour le jeu. Il suffit pour cela de l'ajouter dans le fichier message.xml avec les différents messages traduits. Par souci de manque de temps, nous n'avons pas pu gérer le menu au lancement et le menu de fin de partie.

PLANNING CONSTATE DE NOTRE PROJET

Maintenant que le projet est terminé, voici le planning effectif de ce que nous avons fait (ce planning a été dressé à l'aide de logs de conversations Discord, les dates ne sont pas exactes mais représentent correctement une approximation du début et fin des tâches) :

Du	Au	Tâche
15/09	22/09	Prise de connaissance du jeu (vidéos, parties jouées ensemble)
05/10	06/10	Rédaction du premier rapport intermédiaire
08/10	12/10	Conception UML de l'application
08/10	13/10	Ebauche d'une GUI
14/10	29/10	Développement des composants logiques avec des interactions fonctionnelles
23/11	26/11	Rédaction du deuxième rapport intermédiaire
16/10	07/11	Première version de la GUI
03/11	12/11	Discussions autour de comment implémenter l'IA
08/11	29/11	Version presque finale de la GUI
11/11	26/11	Création de ressources graphiques
27/11	20/12	Développement de l'IA
01/12	08/12	Version finale de la GUI
04/12	06/12	Rédaction du troisième rapport intermédiaire
13/12	-	Développement de la sauvegarde (pas assez de temps pour rattraper la dette technique)
20/12	23/12	Rédaction du rapport final

Pour comparaison, voici le Gantt que nous avions fait au début du projet :

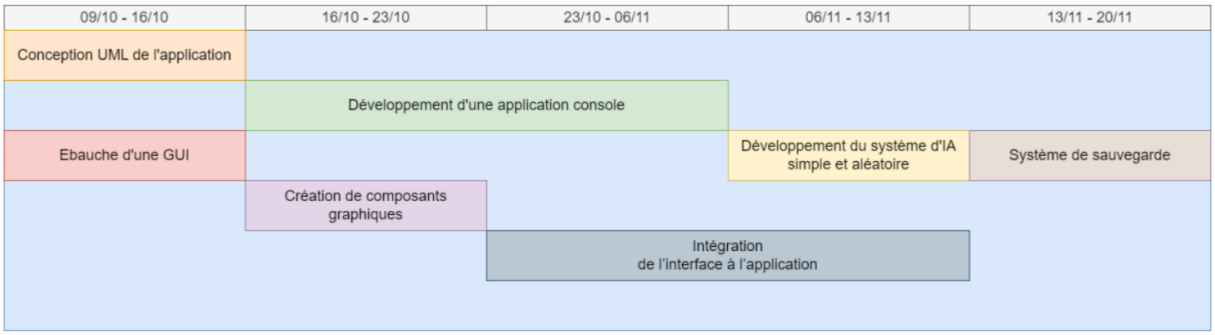


Figure 6 : Gantt créé au début du projet

On remarque directement que nous d'avons finalement pas fait d'application console. En effet, nous développions notre interface graphique en parallèle de nos composants logiques, ce qui nous a permis d'accélérer le développement.

Nous nous sommes plutôt bien tenus à nos prévisions pour le côté logique applicative mais nous avons revu nos ambitions à la hausse pour la GUI et avons donc pris plus de temps à sa conception.

Le développement de l'IA a également pris plus de temps mais cela a été plus la faute a des soucis de conceptions et une vision optimiste de la complexité de celle-ci.

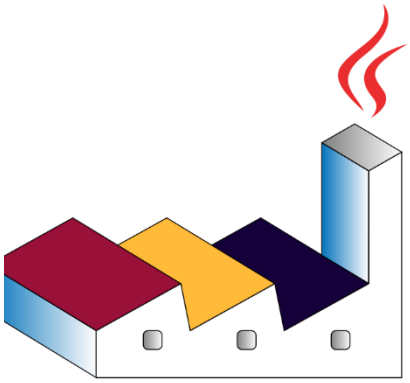
Les problèmes concernant la sauvegarde ont déjà été abordés dans ce rapport. Il s'agit ici des conséquences de la dette technique que nous avons accumulé tout au long du développement.

OUTILS UTILISES

Voici les outils que nous avons le plus utilisé au cours de ce projet :

- Discord : Organisation, communication, dépôt de fichiers légers
- Visual Studio 2022 : Développement de la solution applicative
- Microsoft Word : Rédaction de rapports
- Drive : Partage de fichiers lourds
- GitHub : Dépôt de code, versionnement du code
- IntelliJ et PlantUml : Création de schémas UML

Voici le lien de notre dépôt GitHub si vous souhaitez y voir les branches et commits :
https://github.com/AI22-Splendor/Splendor_DUEL



CONTRIBUTIONS PERSONNELLES DES MEMBRES

Voici un tableau récapitulatif des contributions personnelles des membres de l'équipe :

Membre	Tâches	Part de contribution	Temps passé
Martins Clément	- Conception du jeu - Développement de l'UI - Participation à la logique du jeu - Correction de l'IA - Rédaction de rapports	35%	40h
Bidaux Alexandre	- Conception du jeu - Développement de la logique du jeu - Développement de la sauvegarde (n'a pas pu être implémenté) - Travail sur les ressources graphiques - Rédaction de rapports	30%	30h
Pereira Hugo	- Conception du jeu - Participation à la logique du jeu - Création des ressources graphiques du jeu - Rédaction de rapports	15%	10h
Peter Cyril	- Développement de l'IA - Participation à l'UI - Rédaction de rapports	10%	10h
Hookoom Hans	- Participation au développement des menus - Tests et retours sur l'état de l'application - Rédaction de rapports	10%	10h