Master 1 GIL -
Document d'Architecture
Ecart d'architecture prévions et réalisation
Agora V3-1

Groupe 1 Agora V3-1 : Partie bibliothèque de jeux

14 mai 2024

Version	4
Date	14 mai 2024
Rédigé par	BINGINOT Etienne MAZUY Axelle

Mises à jour du document

Version	Date	Modification réalisée
1	25 février 2024	Création du document
2	4 mars 2024	Ajout des modifications Splendor
3	7 avril 2024	Ajout des modifications Glenmore
4	14 mai 2024	Ajout des modifications Myrmes

Table des matières

1 Objet du document

Ce document a pour but de décrire la différence entre l'architecture logicielle prévue et réalisée du projet Agora pour la partie jeux.

Nous présenterons ainsi la raison de ces changements et ce qu'ils ont entrainé sur l'architecture.

2 Modifications apportées à la DAL générale

2.1 Suppression du GameController

Notre objectif étant la factorisation du code, nous avions prévu d'implanter un contrôleur s'occupant d'afficher le plateau de jeu des différents jeux, ou encore d'autres fonctionnalités comme le rafraichissement automatique de la page une fois en jeu.

Toutefois, lors de la réalisation, nous nous sommes aperçus que ceci n'était pas réalisable, ou du moins pas de manière propre. En effet, chaque jeu nécessite des paramètres différents pour son affichage, ce qui implique également que l'on ne peut pas factoriser sans créer d'énormes structure conditionnelles affichant le jeu en fonction du jeu appelé.

C'est pourquoi nous avons pris la décision de déléguer l'affichage des jeux aux contrôleurs du jeu concerné.

De même pour le rafraîchissement, il n'était pas possible de le factoriser pour tous les jeux puisqu'en fonction de ce qui est rafraichi, la notification Mercure n'est pas la même, les paramètres non plus.

Nous avons ainsi modifié le DAL général afin de ne laisser que la notion globale d'affichage d'un jeu, en expliquant que cette notion, bien que semblant identique pour tous les jeux d'un point de vue architecture, n'est sensiblement pas le même en fonction des jeux. De même, le diagramme de séquence pour les notifications Mercure a été laissé, impliquant également que ce rafraîchissement sera dépendant du contrôleur où il se trouve.

2.2 Renommage du ChatController

Cela n'est pas un grand changement d'architecture, toutefois le contrôleur ChatController a été renommé en MessageController pour un souci de cohérence avec le service et l'entité qui y sont associés.

2.3 Ajout d'un champ à la DTO Player

Un booléen pour le tour du joueur a été ajouté à la DTO (dont la valeur est true si c'est au tour du joueur, false sinon). Ce flag sera utile pour tous les jeux où un tour de joueur à un sens, qu'il s'agisse d'un jeu par phase ou un jeu au tour à tour.

3 Modifications apportées à la DAL du 6 qui prend

3.1 Modification des entités

Le champ chosenCard a été supprimé de la Base de données étant donné qu'il s'agissait davantage d'un champs front-end.

3.2 Modification des droits de regard du spectateur

Après réunion avec le client, les droits de regard du spectateur sur le jeu a été revu pour que le joueur ne puisse pas voir plus que l'ensemble des joueurs présents. Pour le 6 qui prend, cela signifie donc que le spectateur ne voit pas les jeux des joueurs, la gestion a donc été adaptée.

4 Modifications apportées à la DAL du Splendor

4.1 Modification des entités

Le joueur possédant un ensemble de cartes et non une seule carte, le champ cartes du personalBoard est devenu une collection.

De même, une nouvelle entité CardCostSPL a été créée permettant de réunir le champ couleur et le nombre de joyaux pour le coût d'une carte dans DevelopmentCardSPL, de même pour NobleTilesSPL.

4.2 Modification sur les services

4.2.1 Type de retour de getRanking

Dans 6 qui prend et Splendor, on a remplacé le type de retour array d'entier, en array de Player, ce qui permet de renvoyer les joueurs triés dans l'ordre, associant ainsi le joueur en question à sa place dans le classement. Ce qui facilite la lecture des informations notamment pour le front (pour l'affichage du classement on peut récupérer identifiant, pseudo et nombre de points du joueur selon sa place).

4.2.2 Modification des types du paramètre de buyCard

Le paramètre anciennement de type DevelopmentCardSPL a été modifié pour devenir un paramètre de type PlayerCardSPL pour plus de cohérence. En effet, une carte de développement possède les mêmes propriétés peu importe la partie, alors qu'une carte de joueur est spécifique à une partie et possède donc les informations nécessaires.

4.2.3 Fusion de méthodes

Les méthodes reserveDrawnCard et reserveFaceUpCard sont fusionnées pour donner une méthode reserveCard qui permet de réserver une carte de développement, qu'elle vienne de la pioche ou des lignes visibles du plateau.

5 Modifications apportées à la DAL du Glenmore

5.1 Modification des entités

5.1.1 Ajout d'une DTO BoardBoxGLM

Une DTO BoardBoxGLM a été ajoutée afin de contenir des éléments back simplifiant le transfert des données front-back par l'intermédiaire du contrôleur. Cette DTO est notamment utile dans le cas de l'affichage des tuiles sur le plateau personnel.

5.1.2 Ajout d'une entité WarehouseLine

Une entité WarehouseLine a été ajoutée afin de représenter une ligne de l'entrepôt. Cette ligne contient une certaine quantité d'une ressource spécifique (forcément un cube) ainsi qu'une quantité d'argent. Cette modification était nécessaire pour pouvoir initialiser la quantité d'argent de chaque ligne mais également pour qu'en front cela soit plus facile de représenter les données

5.1.3 Ajout d'un champ dans MainBoardGLM

Un champ lastPosition a été ajouté dans MainBoardGLM, afin de stocker en BDD la position du dernier joueur à avoir joué et ainsi calculer le prochain joueur à devoir jouer

5.1.4 Ajout d'un champ selectedTile dans PersonalBoardGLM

Un champ selectedTile a été ajouté dans PersonalBoardGLM représentant une tuile sélectionnée par le joueur lors de la phase d'achat. Une BoardTileGLM, qui est sélectionnée par le joueur, peut donc être achetée (le joueur dispose des ressources), mais la vérification de placement n'a pas encore été faite. S'il peut la placer, la selectedTile devient une PlayerTile (donc avec de vraies coordonnées) Sinon, la selectedTile revient à null et le processus de choix de tuiles recommence.

5.1.5 Ajout de champs dans PlayerTileGLM

3 champs ont été ajoutés dans PlayerTileGLM :
- activated : booléen permettant de savoir si une tuile a déjà été activée lors du tour du joueur ou non
- coordX et coordY : entiers représentant les positions (x, y) d'une PlayerTileGLM sur le plateau personnel. Ces 2 entiers ont été ajoutés pour faciliter la représentation graphique du plateau personnel, pour le front

5.1.6 Ajout du champ player dans PlayerTileResource

Ce champ permet de facilement récupérer la totalité des resources d'un joueur, sans avoir à regarder précisément chacune de ses tuiles.

5.1.7 Ajout d'une DTO PersonalBoardBoxGLM représentant une case du plateau personnel

Ce champ a été ajouté pour calculer les coordonnées des cases vides pour la pose des tuiles sur le plateau personnel.

5.1.8 Ajout du fonctionnement du bot

Le champ isBot dans le playerGLM pour identifier le joueur bot, et ajout d'un champ isDice dans pawn car le pion du bot est un dé.

5.1.9 Ajout d'une entité WarehouseSelectedResourceGLM

Cette entité a été ajoutée afin de permettre de regrouper les ressources d'un joueur qui ont été achetés dans la Warehouse.

5.1.10 Modification de PersonalBoardGLM

Deux champs ont été ajoutés :
— buyingTile : tuile en cours d'achat (BoardTileGLM)
— activatedTile : tuile en cours d'activation (PlayerTileGLM)

5.1.11 Ajout de champs dans PlayerGLM

Deux champs ont été ajoutés :
— roundPhase, indiquant dans quelle phase de jeu le joueur se trouve (phase d'achat, d'activation, de déplacement...)
— activatedSelectionResource, indique si, dans la phase courante, le joueur doit sélectionner des ressources

5.1.12 Ajout d'une DTO pour contenir les informations concernant les ressources d'un joueur

Cette DTO est notamment utile pour l'affichage puisque ce type regroupe les informations essentielles pour les ressources d'un joueur au même endroit.

5.2 Modification sur les services

5.2.1 Ajout d'un service DataManagementService

Un service DataManagementService a été ajouté pour la conversion des données back en données utilisables pour le front, notamment pour le placement des tuiles sur les plateaux.

5.2.2 Découpage de services

Les fonctionnalités ont été découpées en plusieurs services par rapport à ce qui avait été indiqué sur les premières versions de la DAL de Glenmore.

6 Modifications apportées sur la DAL de Myrmes

6.1 Modification des entités

6.1.1 Ajout d'une DTO BoardTileMYR

Une DTO BoardTileMYR a été ajoutée contenant une tuile et indiquant si elle contient le pivot utilisé pour le placement d une pheromone ou non.

6.1.2 Modification de l'entité MainBoardMYR

Les modifications suivantes ont été réalisées :
— Ajout du champ anthillHoles dans le mainBoard pour facilement récupérer l'ensemble des trous
— Ajout du champ tiles dans le mainBoard pour récupérer les tuiles de la partie
— Ajout du champ preys dans le mainBoard pour stocker les proies de la partie sur le plateau
— Ajout d'un champ seasons représentant l'ensemble des saisons de l'année en cours (il n'y a plus d'actualSeason)

6.1.3 Modification de l'entité PlayerMYR

Deux champs ont été ajoutés dans PlayerMYR :
— color pour le player afin de connaître la couleur des éléments associés (la couleur du joueur, qui est notamment utilisée pour le pion de niveau sur la fourmilière, les couleurs des phéromones, etc).
— phase pour connaître la phase du jeu pour le joueur

6.1.4 Ajout d'un champ dans PersonalBoardMYR

Un champ selectedEventLarvaeAmount a été ajouté dans personalBoard. Il a été créé pour pouvoir choisir son bonus d événement et naviguer sur la piste (utile pour les nourrices également).

6.1.5 Ajout d'un champ dans SeasonMYR

Un champ actualSeason booléen indiquant si la saison est celle en cours ou non a été ajouté.

6.1.6 Retrait d'un champ dans NurseMYR

Le champ position a été retiré, puisque son utilité n'est plus avec le champ area qui décrit le même fonctionnement.

6.1.7 Ajout d'un champ dans GameMYR

Le champ gamePhase a été ajouté, permettant de vérifier la phase du jeu actuellement en cours (qui peut être différente de celle des joueurs si ceux-ci ont terminé la phase).

6.2 Modification des services

6.2.1 Découpage des services

Un découpage par saison a été réalisé afin d'isoler certaines fonctionnalités et d'alléger les services (toutefois le découpage peut être amélioré).

6.2.2 Modification des paramètres de placeNurse

Les paramètres ont été modifiés pour la méthode placeNurse, prenant maintenant un joueur et la position souhaitée (qui est un entier).

7 Factorisation et retour sur les jeux

Lors de la phase de factorisation, les DTO et entités ont été modifiées afin de généraliser les comportements similaires.

7.1 Ajout d'un champ score dans la DTO Player

Un champ score représentant les points obtenus par un joueur a été ajouté à la DTO, se faisant, le champ points a disparu des entités Player<GAME> des différents jeux.

7.2 Ajout d'un champ points dans la DTO Card

Un champ points a été ajouté dans la DTO Card représentant les points que porte une carte. De ce fait, le champ prestigePoints a disparu de l'entité CardSPL.

7.3 Modification des fichiers Parameters

Les fichiers <Game>Parameters contenant les paramètres et constantes des différents jeux, ont été passés de classe à interface, et chaque élément s'y trouvant est devenu const (et non plus un attribut public).

8 Tentatives de modifications

8.1 Tentative de refactor des classements des joueurs

On pourrait penser que le refactor des classements simple et sans ambiguïté selon les jeux. En effet on remarque des différences majeures quant aux calculs du score, certains jeux vont avoir un classement décroissant et d'autre le contraire. Pour ces éléments on aurait pu simplifier. Mais certains jeux comme le Splendor demandent des éléments comme le nombre de cartes de développement achetées par un joueur en cas d'égalité des points de prestiges, d'autre comme le Myrmes et le Glenmore vont demander un certains nombre de ressources pour calculer les classements.

De plus, si jamais on faisait un refactor, la grosse partie de la méthode (la totalité) dépend de la fonction de tri qui serait utilisée en fonction du jeu. On ne gagnerait donc pas énormément de lignes, et on aurait un code plus complexe à utiliser.