Projet LO21 : 7 Wonders Duel

Compte-Rendu n°2

Nadji Bensalem
(auteur responsable)

Joshua Nancey
Chloé Taurel
Valentin Ronsseray

Génie Informatique – TD Mercredi matin – Printemps 2024

Table des Matières

I Révision du modèle conceptuel

Une mise à jour du modèle conceptuel a été opérée. La figure I.1 représente le nouvel UML de notre projet. Les attributs et les méthodes ne sont pas explicités.

Nous avons décidé d'améliorer notre modèle conceptuel pour les raisons suivantes :

- D'une part, l'exemple du SET! vu dernièrement en TD nous a inspiré à créer un contrôleur dans notre jeu, représenté par la classe **Partie**.
- D'autre part, la notion d'héritage intervient naturellement dans le projet puisque certains objets du jeu concernent de mêmes sous-ensembles (*e.g.* les cartes, bâtiments, plateaux etc).
- Enfin, les relations d'agrégation et de composition permettent de préciser davantage la nature des associations. En particulier, l'utilisation d'une composition entre la classe **Cartes** et la classe **Jeu** illustre le fait que le **Jeu** est responsable du cycle de vie des **Cartes**.

II Remarques

II.1 Polymorphisme

Nous prévoyons de faire évoluer et d'améliorer l'implémentation en maîtrisant davantage les concepts d'héritage et de polymorphisme. Notamment, nous nous servons à plusieurs reprises de tableaux de pointeurs de type **Bâtiment** afin de regrouper tous les **bâtiments** possédés par un **Joueur**. Notre maîtrise du polymorphisme étant toujours limitée, nous avons considéré dans certaines parties du code qu'un **Joueur** possédait un tableau différent pour chaque type de **bâtiment** afin de pouvoir accéder à leurs attributs et méthodes. Nous souhaitons par la suite stocker tous les **bâtiments** dans un même tableau afin de rendre le code plus clair et plus adaptable aux changements.

II.2 Jetons Progrès

Nous n'avons pas encore implémenté les particularités que les jetons progrès apportent au jeu. Nous avons néanmoins réfléchi à la manière de les prendre en considération. Dans notre premier rapport, nous avions mis une méthode pour chaque jeton dans la classe Jeton, mais après réflexion il sera plus pertinent d'adapter les méthodes déjà implémentées des autres classes aux particularités des jetons:

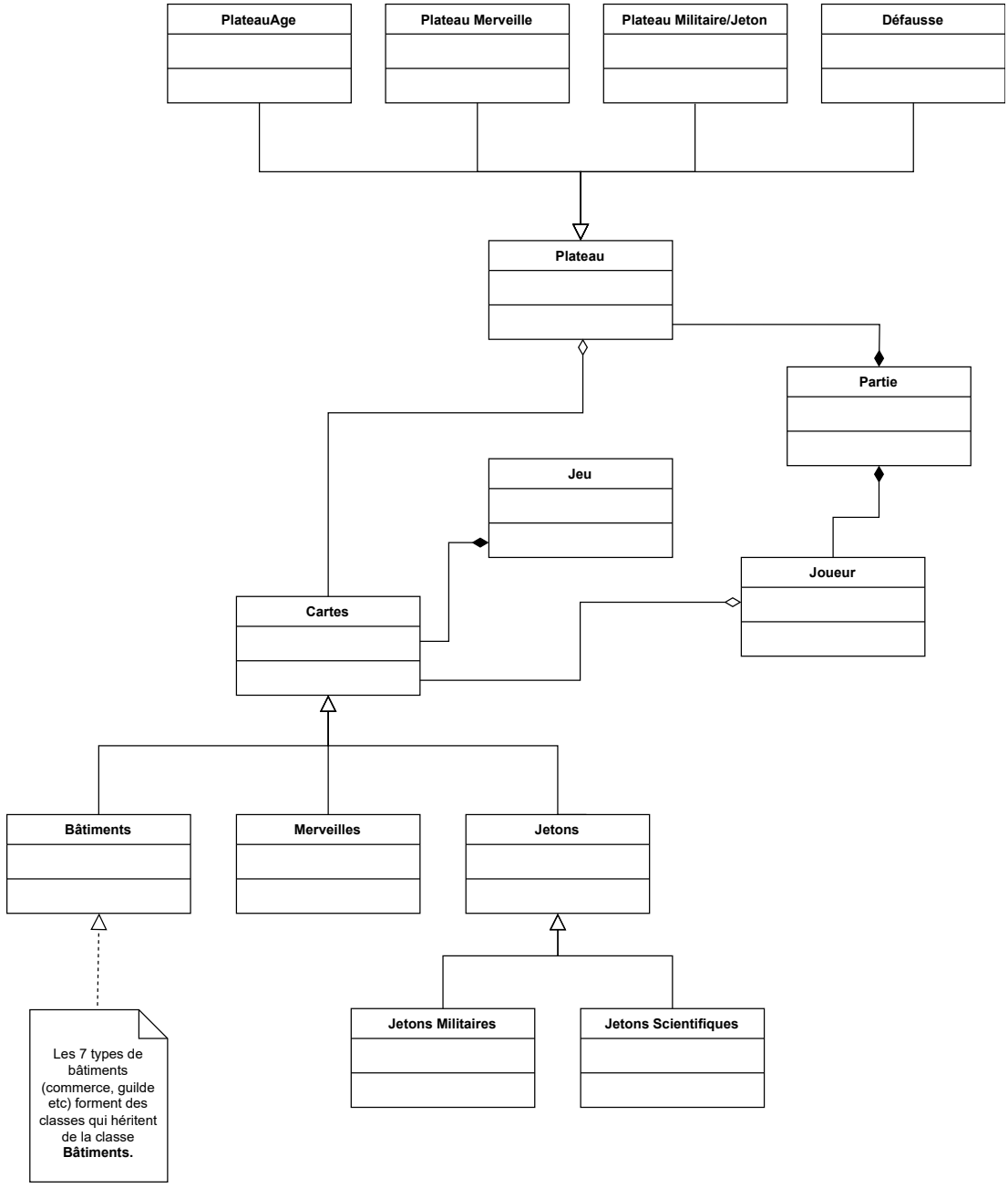

Figure I.1: Modèle UML

Maçonnerie : “Les prochaines cartes bleues construites par le joueur lui coûteront 2 ressources de moins. À chaque construction, le joueur est libre de choisir sur quelles ressources porte la réduction.”

La classe bâtiment doit s’en occuper. Elle doit modifier la valeur renvoyée par `prix_final` si le bâtiment est un bâtiment civil.

Autre option : la classe `Joueur` s’en occupe, dans la méthode Action en baissant la valeur renvoyée par `prix_final` si le joueur possède le jeton Maçonnerie.

Mathématiques : “À la fin de la partie, le joueur marque 3 points de victoire pour chaque jeton Progrès en sa possession. Ce Progrès rapporte lui-même 3 points de victoire.”

La classe `Partie` doit s’en occuper. Elle teste si un des joueurs possède le jeton dans sa méthode `victoire_civile` et ajoute 3 points pour chaque jeton progrès en sa possession s’il possède Mathématiques.

Philosophie : “Le jeton rapporte 7 points de victoire.”

La classe `Joueur` s’en occupe, dans sa méthode ajouter_jeton. Si Philosphie est choisie, on ajoute 7 points de victoire à l’attribut points du Joueur.

Agriculture : “Immédiatement 6 pièces + 4 points de victoire.”

La classe **Joueur** s’en occupe dans la méthode `ajouter_jeton`. Met à jour attributs solde et points du joueur.

Architecture : “Les prochaines Merveilles construites par le joueur lui coûteront 2 ressources de moins. À chaque construction, le joueur est libre de choisir sur quelles ressources porte la réduction.”

La classe `Bâtiment` s’en occupe, dans sa méthode `prix_final()` en vérifiant si le Joueur possède ce jeton.

Autre option: la classe `Partie` s’en occupe en laissant le Joueur choisir deux ressources à utiliser et modifiant la valeur renvoyée par `prix_final()`.

Économie : “Le joueur récupère l’argent dépensé par son adversaire lorsqu’il achète des ressources.”

La classe Bâtiment et Partie s’en occupent. Quand un joueur construit un bâtiment, la partie appellera une méthode de la classe bâtiment servant à déterminer l’argent dépensé dans les ressources seulement. La classe Partie ajoutera cette somme au solde du joueur qui n’a pas construit le bâtiment (c’est-à-dire son adversaire).

Loi : “Ce jeton rapporte un symbole scientifique.”

Simple. La classe Joueur s’en occupe dans la méthode choix_jeton.

Stratégie : “À partir du moment où le joueur possède la Stratégie, les prochaines cartes rouges qu’il construit bénéficieront de 1 bouclier supplémentaire.”

La classe `Partie` s’en occupe. Quand un bâtiment militaire est construit, elle vérifie si le joueur possède ce jeton ou non et modifie le solde militaire en conséquence.

Théologie : “Les prochaines Merveilles construites par le joueur sont considérées comme ayant toutes l’effet ” Rejouer ”.”

La classe `Partie` s’en occupe à la construction d’une merveille. Elle vérifiera si le joueur qui construit une merveille possède ce jeton et le fera rejouer si c’est le cas (pas de modification si la merveille a déjà l’effet rejouer).

Urbanisme : “Le joueur prend immédiatement 6 pièces de la banque. Chaque fois que le joueur construit un Bâtiment gratuitement par chaînage il reçoit 4 pièces.”

La classe joueur s’occupe de l’effet immédiat dans sa méthode `choisir_jeton()`. Quand un joueur choisit de construire un bâtiment, la classe contrôleur Partie teste s’il possède ce jeton. Si oui, elle teste aussi si le bâtiment a été construit avec chaînage grâce

à la méthode `possede_chainage()` de Joueur. Si les deux conditions sont réunies, alors Partie met à jour le solde du joueur.

III Participations au projet

III.1 Valentin

Tâches accomplies :

- Contribution avec le groupe sur la formation du nouvel UML. *Durée estimée : 1h30*
- Création du nouvel UML sous format numérique. *Durée estimée : 1h*
- Prise en main de l'environnement Qt pour plus tard. Le framework Qt, permettant de pouvoir concevoir l'interface graphique pour le jeu, sera utilisé dès lors que le code compilera. *Durée estimée : 3h*
- Rédaction du rapport. *Durée estimée : 3h*
- Contributions sur le code :
  - Constitution du singleton **Jeu** qui gère le cycle de vie des **Cartes**. *Durée estimée : 1h*
  - Constitution de la classe **Plateau**. *Durée estimée : 2h*
  - Création d'une fonction utilitaire qui construit un vecteur d'entiers choisis aléatoirement. Elle permettra de gérer la plupart des aspects aléatoire du jeu. *Durée estimée : 1h*
  - Autres contributions mineures sur l'ensemble du code, notamment sur la correction des programmes et l'ajout d'accesseurs et de méthodes simples. *Durée estimée : 3h*

Temps de travail estimé pour le rapport 2 : 15 heures. Temps de travail estimé pour le rapport 2 : 12 heures.

Temps de travail total estimé : 27 heures.

III.2 Chloé

Tâches accomplies :

- Contribution avec le groupe sur la formation du nouvel UML. *Durée estimée: 1h30*
- Implémentation de la classe **Joueur** et méthodes associées. Implémentation d'autres méthodes avant que nous changions d'UML et que nous considérions que la classe **Partie** serait le contrôleur. Temps passé notamment à réfléchir à l'implémentation des méthodes de sélection. *Durée estimée: 5 heures*
- Implémentation du constructeur de **PlateauAge** avec la distribution aléatoire des cartes en début de partie, à partir de l'ensemble du jeu (piocher dans le singleton Jeu). *Durée estimée: 3 heures*

- Autres contributions mineures sur l'ensemble du code, notamment sur la correction des programmes. *Durée estimée : 2h*
- Rédaction de la partie Remarques et réflexion derrière les méthodes pour les jetons progrès *Durée estimée: 1h30*
- Redaction du rapport *Durée estimée: 2 heures*

Temps de travail estimé : 15h. (+15 heures sur le rapport 1)
Temps total estimé à 30 heures.

III.3 Joshua

Tâche accomplies :

- Révision de la classe **Carte** et de ses filles, notamment certains attributs comme les ressources. *Durée estimée : 2h*
- Révision de la classe **Joueur** et implémentations des nouveaux tableaux filles de bâtiment.*Durée estimée : 2h*
- Le principal de mon travail à été de réaliser les cartes ainsi que toutes les liens que ces dernières ont avec le jeux voici plus de détail et mon impact sur le code :
  - Création de toutes les classes **Cartes**, **Bâtiment**, **Merveilles**, et ainsi que tout les types de bâtiments en utilisant l'héritage.*Durée estimée : 2h30*
  - Création du fichier `carte.cpp` et de la méthode `prix_final` pour les deux joueurs, prenant en compte la méthode `est_chainée`. *Durée estimée : 2h*
  - Rédaction de tous les accesseurs et constructeurs de tous les types de **Cartes**. *Durée estimée : 2h*
  - Création du fichier `batiment.cpp` possèdant les constructeurs spéciaux merveilles et commerce.*Durée estimée : 2h*
  - Implémentation de la méthode `est_chainée` dans **bâtiment** pour savoir si un bâtiment est chaînée avec des bâtiments d'un joueur.*Durée estimée : 1h*
  - Redaction du rapport *Durée estimée: 2 heures*

Temps de travail estimé pour le rapport 2 : 15 heures.(+9h sur le rapport numéro 1) Temps de travail total estimé : 24 heures.

III.4 Nadji

Tâches accomplies :

- Remaniement de modèle conceptuel (UML). *Durée estimée: 3 heures*
- Élaboration et implémentation de la classe **Partie** et ses méthodes. Réflexion à la conception de cette classe, pour gagner en efficacité à l'instar de la classe contrôleur de l'exemple Set! vu précédemment en TD. *Durée estimée: 3 heures*

- Implémentation de la classe **Jeton** et de ses méthodes, et contribution avec le groupe à opérer des changements inetrvenants dans certaines avec l'ajout et la suppression d'attributs de méthodes avant la remodélisation du modèle conceptuel. *Durée estimée: 4 heures*

- Autres contributions mineures sur l'ensemble du code, notamment sur la correction des programmes et ajout d'accesseurs en lecture nécessaire à certaines méthodes. *Durée estimée : 2h30*

- Rédaction du rapport *Durée estimée: 2 heure*

Temps de travail estimé : 15 heures. (+6 heures sur le rapport 1)
Temps total estimé à 21 heures.

IV Affectation des tâches restantes

IV.1 Chloé

- Modifier le code avec des techniques de polymorphisme pour permettre de stocker différents types de batiments dans un pointeur `Batiment**` tout en gardant l'accès aux attributs spécifiques des différents types. Cette tâche sera effectuée à deux avec Joshua puisque les classes **Joueur** et **Batiment** dont nous nous sommes occupés sont les premières concernées par les éventuels changements engendrés.

- Terminer les méthodes de la classe Joueur. *Durée estimée: 1heure*

- Méthodes relatives aux jetons. (voir partie Remarques) *Durée estimée: 4 heures*

- Méthodes relatives aux cartes de l'âge 3 ayant pour effets un changement de statut du joueur (solde, points de victoire) à la fin du jeu ou immédiatement.*Durée estimée: 5 heures*

- Fonction de sélection des merveilles en début de partie. Il faut que les joueurs puissent à tour de rôle choisir leurs merveilles et qu'elles soient placées dans leur cité. *Durée estimée: 4 heures.*

- Ajouter et implémenter les méthodes dont a besoin la classe **Plateau**. Il faut une méthode `supprimerCarte()` qui renvoie si cette carte était la dernière afin de savoir si c'est la fin d'un **Âge**. *Durée estimée: 3 heures.*

- Réfléchir à la pertinence de l'architecture pour les extensions Agora et Panthéon. Proposer modifications éventuelles. *Durée estimée : 2 heures.*

IV.2 Joshua

- Fonction de sélection du choix d'un **Joueur**. Cette fonction devra être dans la classe **Partie**, et devra impliquer l'interface graphique afin de permettre le choix. Elle devra utiliser toutes les fonctions implémentées jusqu'ici afin de vérifier qu'un **joueur** a les ressources et solde nécessaire pour faire un choix (construire une merveille, un bâtiment). *Durée estimée: 15 heures.*

- Création de toutes les cartes par le constructeur du singleton Jeu. Également destruction de toutes ces cartes dans le destructeur de Jeu. *Durée estimée: 6 heures.*

IV.3 Valentin

- Participer à la rédaction de la partie où les différentes cartes seront instanciées dans le **Jeu**. En effet, cette partie est un défi majeur car contrairement aux SET! où les cartes se caractérisaient par leurs 4 attributs (et donc il suffisaient de créer 4 boucles `for` imbriquées), les cartes de nos jeux ont leurs propres spécificités ce qui nous oblige à les créer manuellement une par une.
- Contribuer de manière globale au code présent avant l'implémentation de l'interface graphique. Il faudrait essayer de faire en sorte que l'on puisse jouer au jeu, au moins en mode console pour la prochaine échéance.
- Réfléchir brièvement à notre interface graphique : comment seront construit les différents menus ? A quoi ressemblera le plateau de jeu ? Quelles sont les connexions entre les menus etc.
- Affichage des cartes (compréhensible visuellement). Création de widgets pour pouvoir cliquer dessus. *Durée estimée: 10 heures*

IV.4 Nadji

- Terminer les méthodes manquante de la classe partie,comme victoire civil,etc..*Durée estimée: 4 heures*
- Réfléchir à la pertinence de l'architecture pour l'ajout de différents types d'IA. *Durée estimée: 2 heures.*
- Affichage des plateaux, en faisant attention à la disposition des Cartes selon l'âge, ainsi qu'à leur disposition (face cachée, accessibilité,etc) *Durée estimée: 10 heures*
- Développer une IA de bas niveau (aléatoire). *Durée estimée: 3 heures.*