Michael FERNANDEZ - GI02
Soulaymane KEBLI - GI02
Florian MIOUSSET - TC03
Tom PERDEREAU - GI01
Mohamed TAHIRI - GI02

Université de Technologie de Compiègne

LO21 - Programmation et conception orientées objet

Responsable et Chargé de TD : Antoine JOUGLET

Rendu 2 - 7 Wonders Duel

TD4 - Groupe n°3

Sommaire

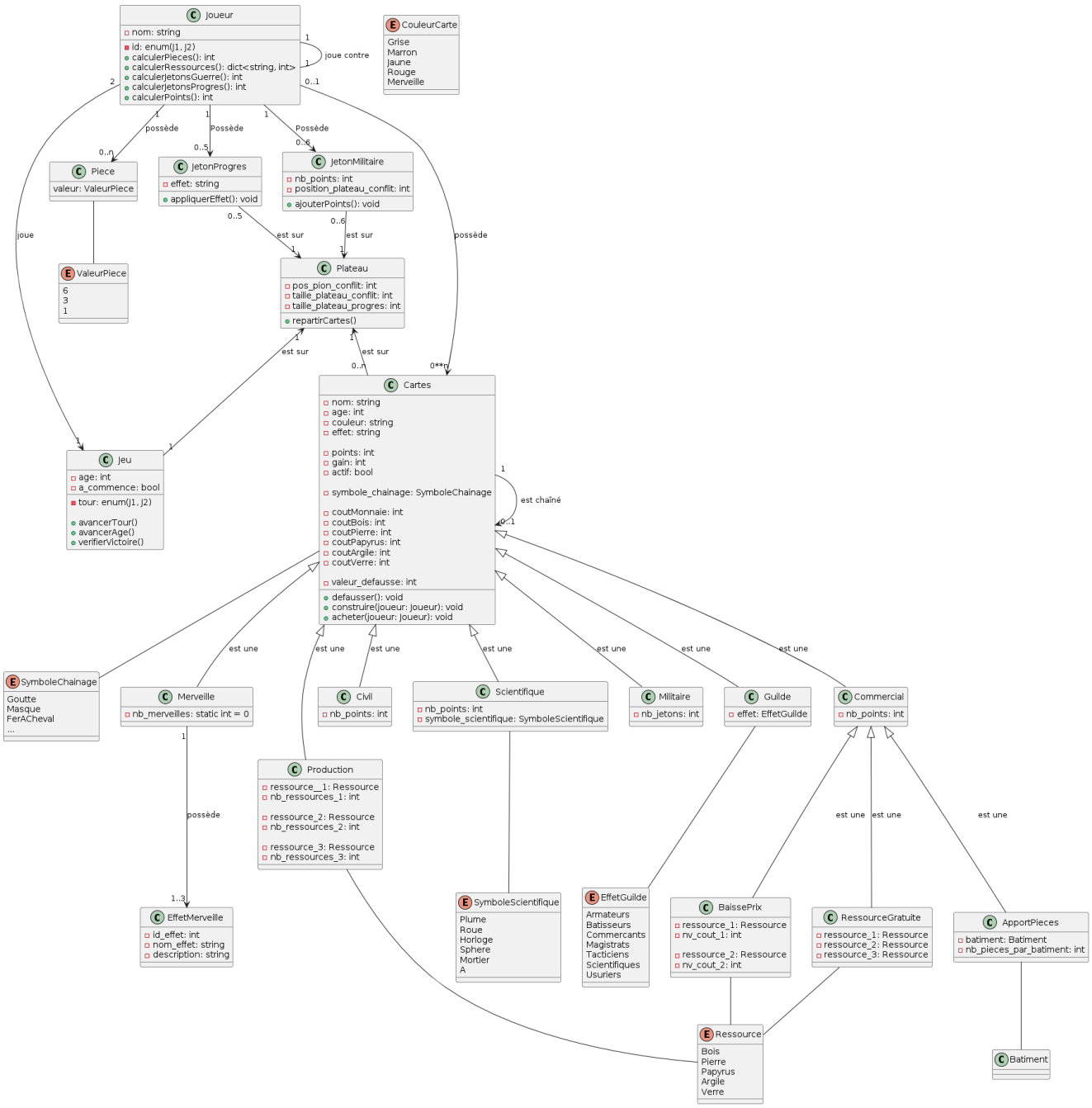
1 Introduction

Dans le dernier rapport, nous présentions le jeu 7 Wonders Duel, son fonctionnement, les différentes cartes le composant ainsi que les différentes façons de gagner. Nous proposions également en page 8 un MCD UML, qui nous a permis de proposer une version préliminaire de l'architecture que nous pensions implémenter dans le projet.

Ce rapport tâchera de présenter une nouvelle version de l'architecture envisagée, une ré-actualisation des tâches à effectuer ainsi que le bilan de cohésion d'équipe du mois passé. Le compte rendu d'une réflexion que nous avons eue sur les design patterns qui pourraient être intéressants à implémenter sera également donné.

2 Évolution d'Architecture (UML)

Nous avons repensé l'architecture de notre projet en la précisant. Cette précision se trouve à deux niveau : premièrement, la création de nouvelles classes pour déléguer certaines fonctionnalités (notamment Jeu, mais aussi Commercial pour gérer le côté financier des cartes), ainsi que des énumérations pour commencer à se rendre compte de la diversité des cartes qu'il sera nécessaire d'implémenter, en profondeur. Mais aussi l'ajout de méthodes qui créent du lien entre les différentes classes, ce qui manquait dans l'UML préliminaire et introductif.



Redirection vers plantuml.com pour une version agrandie de l'UML https://shorturl.at/fsuI4

3 Architecture : Implémentation et Possibilités

En réfléchissant à l'évolution de l'architecture de notre projet, en ayant vu les design patterns en cours et donc en se penchant sur l'avant dernier point du sujet, nous avons entamé une réflexion sur l'usage que nous pourrions faire des design patterns dans notre projet, et donc en quoi ceux-ci nous aideraient à concevoir notre architecture.

3.1 Design Pattern : Template Method

L'utilisation du design pattern Template Method permettrait de gérer d'une manière plus propre et mieux adaptée la gestion de l'Héritage entre les différents types de carte. Au niveau du comportement de celles-ci. Nous aurions alors une classe mère 'Carte' abstraite dont tous les différents types de cartes hériteraient. En effet, toutes les cartes possèdent une même base commune (y compris du point de vue des methodes) quels que soit leur type : un coût de création, un nom, une méthode pour la 'piocher' et la mettre dans le jeu/sur le plateau, une méthode de défausse... Cependant, lorsque l'on entre dans les détails, en fonction de leur type, des méthodes propres vont être nécessaire : produire des ressources pour les cartes matières premières, les produits manufacturés ; donner des points de victoire pour les bâtiments civils ; obtenir des jetons scientifiques pour les bâtiments scientifiques ou encore modifier les règles d'achats des ressources avec les bâtiments commerciaux ; augmenter la puissance militaire pour les bâtiments militaires et gagner des points en fonctions de critères avec les guildes. Les cartes merveilles possèdent aussi leurs méthodes propres, en fonction des effets. Enfin l'un des points majeurs poussant à l'utilisation du design pattern Template Method est l'utilisation des opérations 'Hook'. Les opérations 'Hook' permettent alors l'implémentation d'extension pour des nouvelles cartes ayant des fonctionnalités divergentes de l'utilisation initial des objets 'cartes'. Ces opérations peuvent aussi être utile en fonction des états/ du statut des cartes au sein du jeu : redéfinir une méthode afin de la nullifier lorsqu'une carte est défaussée.

3.2 Strategy

Le design pattern 'Strategy' est propice à l'insertion de niveaux de difficultés pour L'IA au sein du jeux. Les différents comportements/ les stratégies à utiliser pourront varier en fonction du niveaux de difficulté saisit par l'utilisateur. En effet l'IA de doit pas avoir la même stratégie si elle est en 'facile' que en 'difficile'. Le design pattern 'Strategy' est aussi utile dans le cas où l'utilisateur renseignerait une extension (comme Agora ou Panthéon qu'il veut inclure dans le jeu). Dans ce cas, la mise en place de la partie ainsi que le(s) plateau(x) de jeu est(sont) différent(s). Le déroulement et les conditions de victoires sont aussi différentes/ de nouvelles existent. Pour faire bref, le comportement de la partie globale n'est pas le même en présence d'une extension.

3.3 Composite

Le design pattern 'Composite' aurait pu être utile pour la gestion des différents 'Âges' au sein du jeu. Nous aurions pu séparer au point de vue structurel les différentes cartes en fonction de l'appartenance à un âge (composition). L'utilisation de 'Composite' permettrait alors de répondre parfaitement à ce besoin ; que ce soit pour la séparation en différents tours de jeu (1 par Âge), pour la distribution des cartes au sein de ceux-ci ; ou bien même pour les phases 'd'évolution' des cartes/ des chaînes de cartes à travers les différents âges.

Ainsi nous pourront gérer les cartes appartenant à un Âge de manière uniforme, en ajouter dans le cas d'une extension ; tout en permettant de garder une gestion des cartes uniques pour leurs spécificités. Cependant nous avons décidé de ne pas utilisé ce design pattern pour le moment. Nous implementons les âges comme un simple attribut de classe. L'implementation d'une composition entre les cartes et les âges auraient sous entendu que lors du changement d'un âge ; de la fin de vie d'un âge, toutes les cartes appartenant à celui-ci auraient alors été détruite. Or un joueur est en mesure de créer des bâtiments/de posseder des cartes qui auront des actions tout au long du jeu ; quel que soit l'âge auquel elles appartiennent et l'âge en cours.

3.4 Abstract factory

L'utilisation du design pattern 'abstract factory' est l'un des éléments primordial à la création des cartes pour le jeu. En effet, son implémentation permet alors de créer toutes les cartes en se basant sur une classe mère 'carte' alors abstraite et en créant des classes filles répondant au besoin de chaque type de cartes. L'utilisation de 'abstract factory' est aussi pertinente pour la création d'items pour le fonctionnement de l'interface graphique du jeu ; tant du point de vue des cartes, que des différents jetons de jeu par exemple. Il faut alors creer les différents widgets qui répondront à ce besoin.

3.5 Singleton

Le plateau de jeu est une entité centrale qui contient toutes les informations sur l'état actuel du jeu, telles que les cartes disponibles, les ressources, les points de victoire, etc... Le fait d'utiliser le design pattern Singleton pour instancier le plateau de jeu peut présenter plusieurs avantages comme par exemple l'instanciation unique du plateau pour éviter des potentiels problèmes de synchronisation liés à la gestion de plusieurs instances. Cela permettrait aussi d'avoir un accès global au plateau sans avoir à se soucier des problèmes d'accessibilité des joueurs. De plus, le fait que le plateau soit géré par un singleton permet de faciliter l'évolution du jeu comme par exemple l'ajout ou la suppression de cartes, la mise à jour des ressources etc.. et tout ça grâce aux méthodes fournies par le singleton. Cela facilitera aussi par ailleurs la fonctionnalité d'extension et de maintenance si besoin (comme par exemple le fait d'ajouter l'extension Agora ou Panthéon)

3.6 Iterator

Le design pattern iterator peut être utile pour parcourir les cartes qui sont placées sur le plateau. Cela permettrait de garantir une cohérence dans le traitement de chaque emplacement. Il permettrait aussi de créer une itération personnalisée qui serait utile pour parcourir les cartes d'un joueur spécifique dans un ordre précis.(comme par exemple pour déterminer la victoire scientifique ou permettre la gestion des jetons de progrès scientifique)

Pour la victoire militaire aussi cela pourrait être intéressant comme pour parcourir les différentes étapes du conflit militaire pour déterminer la victoire ou non.

3.7 Mediator

Le design pattern mediator serait utile en ce qui concerne la gestion globale de la partie et son déroulement. Le mediator peut servir à coordonner les différentes phases de jeu(âge), telles que la phase de sélection des cartes, la phase de construction, la phase de conflit militaire. Chaque phase peut être représentée par un composant distinct, et le

mediator orchestre le passage d'une phase à l'autre en fonction des actions des joueurs. Il est aussi utile pour gérer l'interface graphique par rapport au fonctionnement du jeu Cela faciliterai aussi la communication entre les deux joueurs comme par exemple lorsqu'une action affecte l'autre joueur(déclenchement conflit militaire etc..), cela peut se faire par l'intermédiaire du mediator. Cela faciliterait aussi la "communication" entre les cartes du jeu et les ressources comme par exemple lorsqu'un joueur construit une merveille ou une structure et que les ressources doivent être actualisés.

4 Tâches à effectuer et temps prévisionnel par tâche

Voici la répartition ré-évaluée des tâches, avec, en italique : les nouveaux temps de travail, l'état d'avancement des tâches (en pourcentage et en termes d'heures passées), les nouvelles tâches, ainsi que les changements d'affectation des tâches.

- *Etablissement de l'architecture du projet, notamment en pensant aux designs patterns qu'il faudra implémenter* ⇒ *tout le monde (10h) Tâche terminée à 75% : nous avons grandement complété notre UML depuis le dernier rapport, mais nous nous attendons à ce que l'architecture change encore en continuant à voir de nouvelles notions en cours.*
- Discussions et décisions sur les bonnes pratiques de gestion du Github ⇒ tout le monde (5h) *Tâche terminée : nous avons décidé d'adopter la méthode* ⇒ *une branche par issue, que l'on commit dans une branche permettant de tester l'interopérabilité de nos codes, et enfin un commit dans la branche principale. L'estimation temporelle était pertinente.*
- Assurer l'interopérabilité des codes ⇒ tout le monde (10h/personne) *Tâche pas encore commencée car chacun s'occupe premièrement de sa propre partie.*
- Testing ⇒ tout le monde, en fin de projet *Tâche pas encore commencée car il est encore trop tôt.*
- Gestion des classes pour les cartes ⇒ Soulaymane (30h) *Tâche pas encore commencée car il était nécessaire de produire un UML le plus complet possible pour mettre en avant les différents attributs et méthodes.*
- Méthodes de gestion des finances (pièces du joueur, classe pièce, ressources des joueurs, échanges avec la banque) ⇒ Florian (40h) *10% : début de création des classes en fonction de l'architecture retenue, et réflexion sur les liens qu'il sera nécessaire de faire avec les autres classes.*
- Interface graphique, front end (plateau, pions) avec Qt ⇒ Florian & Soulaymane (40h) *Tâche pas encore commencée car il est encore trop tôt. Il semble assez probable qu'il faille plus de 40h pour faire cette interface, cela sera plus clair lorsque nous aurons plus avancé dans le projet.*
- Méthode de gestion de la partie militaire des cartes ⇒ Michaël (10h) *Une méthode consisterait à obtenir l'état de l'avancée du conflit militaire et de stopper la partie si l'une des cases "capitale" est atteinte par l'un des joueurs. Quant à l'architecture concrète, cela pourra être géré par le design pattern Iterator pour calculer efficacement et éviter des problèmes de synchronisation. L'implémentation se fera une fois les classes créés.*
- Méthode de gestion de la partie scientifique des cartes ⇒ Soulaymane (10h) *Les méthodes pour cette tâche ont été partiellement indiquées sur l'UML, l'étape suivante est donc de coder ces différentes fonctionnalités.*
- Méthode de gestion de la partie civile des cartes ⇒ Michaël (10h) *Il serait intéressant d'utiliser des méthodes "get" pour chacun des deux joueurs afin d'obtenir tous leur points de manière séparée et pouvoir en faire la somme, en s'aidant du design pattern Mediator qui pourrait permettre de gérer de telles instances. Si une égalité venait à se produire à la fin de cette somme, il serait alors nécessaire de créer une autre méthode qui s'appuierait cette fois-ci sur le design pattern Iterator afin de déterminer qui a le plus de batiments civils parmi les deux joueurs. L'implémentation est en cours.*

— Gestion de la classe joueur et des méthodes associées ⇒ Tom (20h) *L'implémentation de la classe est en cours : les attributs et les premières méthodes d'accès à ceux-ci sont créées.*

— Automatisation du jeu (distribution initiale des cartes, vérifier si les joueurs ont assez de ressources) ⇒ Mohamed (30h) *Les méthodes pour cette tâche ont été partiellement indiquées sur l'UML, l'étape suivante est donc de coder ces différentes fonctionnalités.*

— Gestion du temps (âge, numéro de tour) ⇒ Mohamed (30h) *Cette tâche a été commencée en intégrant des choix d'architecture adaptés dans l'UML.*

— Intelligence Artificielle basée sur l'aléatoire, avec niveaux d'adversité ⇒ Tom (20h, voire + si plusieurs niveaux de difficultés) *La tâche n'as pas totalement débuté, plusieurs solutions d'implémentation on été réfléchi et celle de l'utilisation du design pattern strategy est la plus pertinente pour le moment. Pour ce qui est de(s) stratégies utilisées par les IA(s), celle-ci débutera sous peu, en parallèle de l'évolution du cours de IA02 (suivie en parallèle de LO21).*

— Vérification des choix d'architecture pour assurer la possibilité d'ajout d'extensions (Agora ou Panthéon) ⇒ Mohamed (5h) *Cette tâche est en continu au cours du projet, l'étape d'analyse des extensions est pratiquement finie.*

Soit ≈ 60h/personne (≈ 300h cumulées) sur la durée totale du projet.

5 Bilan sur la cohésion d'équipe

La conciliation des médians et du projet a été particulièrement ardue, notamment pour trouver des créneaux pour faire des appels de groupe. Néanmoins, à peine les médians finis pour tout le monde, nous nous sommes tout de suite réunis pour mettre en commun d'une manière plus formelle et structurée nos avancées au cours des dernières semaines, ainsi que pour rédiger collectivement ce rapport. Les deadlines que nous nous sommes fixées, suffisamment à l'avance pour ne pas occasionner trop de stress, ont été respectées. Cela nous permet de garder une bonne organisation, qui est efficace et agréable.

Les échanges et suggestions par discussion textuelles sont dynamiques. Cela permet à chacun d'entre nous de détecter certaines choses que nous avons faites qui sont fausses, ou bien qui peuvent être améliorées, et ce de manière très rapide et sans perte de temps.

D'un commun accord, nous allons profiter de la semaine de vacances à venir pour donner un coup de fouet à notre avancée commune sur le projet. Nous prévoyons par ailleurs quelques appels complétés d'échanges textuels, pour s'entraider et garder une dynamique de groupe constructive et effective.

La mise en place du Github, l'établissement d'une architecture bien plus exhaustive qu'avant, ainsi que l'accord collectif des méthodes que nous allons employer pour implémenter en C++ cette architecture, rend le projet plus tangible, concret et donc motivant. Aussi, nous arrivons à y voir bien plus clair sur ce que représente un tel projet grâce aux trois derniers TDs dans lesquels il serait possible de dire que nous avons fait une version (très) réduite du projet.