Projet : *7 Wonders Duel*

LO21 – Compte-Rendu n°1 du Projet

Valentin Ronsseray
(auteur responsable)

Joshua Nancey
Chloé Taurel
Nadji Bensalem

Génie Informatique – TD Mercredi matin – Printemps 2024

Table des Matières

I Introduction

Notre projet consiste à concevoir et à développer d'une application dédiée au jeu de société 7 Wonders Duel. Dans le cadre de ce projet, notre équipe d'étudiants s'est engagée à concevoir et à réaliser une plate-forme numérique qui permettra aux joueurs de profiter pleinement de l'expérience de ce jeu de stratégie. Pour ce faire, nous mettons en œuvre les concepts et les outils basés sur la programmation et la conception orientées objet. Par la suite, nous utiliserons le langage de programmation C++ pour implémenter la structure présentée dans ce rapport, et l'interface graphique sera réalisée à l'aide du framework Qt.

Notre objectif principal est de créer une application conviviale et intuitive qui capture fidèlement l'essence du jeu de société 7 Wonders Duel. Nous visons à offrir une expérience de jeu fluide, interactive et immersive, tout en préservant les mécanismes stratégiques et les choix tactiques qui font le succès de ce jeu. De plus, on utilise le principe d'encapsulation de telle sorte que des changements et des améliorations puissent être apportés par des utilisateurs extérieurs. Et ces changements ne remettrons pas en question le code de base.

Dans ce premier compte rendu, nous livrons notre première analyse des différents concepts qui apparaissent dans le jeu et la structure que nous avons trouvé la plus judicieuse afin de mener à bien le projet. Une présentation du jeu est primordiale, elle met en lumière la constitution du jeu et les axes principaux de son déroulement.

I.1 Préparation du Jeu

Le pion Conflit est placé au centre du plateau sur la case neutre, tandis que les 4 jetons Militaires sont disposés face visible sur leurs emplacements. Ensuite, 5 jetons Progrès sont choisis aléatoirement et révélés sur le plateau, les autres sont remis dans la boîte. Chaque joueur prend 7 pièces à la banque.

I.2 Sélection des Merveilles

Les 12 cartes Merveille sont mélangées et 4 sont placées aléatoirement face visible. Les joueurs alternent pour choisir leurs Merveilles, le premier joueur sélectionne une carte, puis le deuxième en choisit deux, et ainsi de suite jusqu'à ce que chaque joueur ait trois Merveilles.

I.3 Paquets de Cartes par Âge

Trois cartes de chaque paquet d'âge sont retirées sans être consultées et remises dans la boîte. Pour l'âge III, trois cartes Guilde sont ajoutées aléatoirement au paquet, les autres cartes Guildes retournent dans la boîte.

I.4 Déroulement d'une Partie

La partie se déroule sur trois Âges, débutant par l'âge I et se terminant par l'âge III. Chaque joueur choisit une carte "accessible" à tour de rôle. La partie se conclut en cas de suprématie militaire ou scientifique ou à la fin de l'âge III.

I.5 Jouer une Carte

Les joueurs peuvent choisir de construire un Bâtiment, défausser une carte pour obtenir des pièces, ou construire une Merveille en payant son coût.

I.6 Règle du Commerce

En cas de manque de ressources, les joueurs peuvent acheter les ressources manquantes à la banque. Le coût est déterminé par le nombre de symboles de la même ressource produite par les cartes marron et grises de la cité adverse.

I.7 Fin de Partie

La partie se termine immédiatement en cas de suprématie militaire, scientifique, ou à la fin de l'âge III. Les points de victoire sont comptabilisés et le joueur avec le score le plus élevé remporte la victoire civile. En cas d'égalité, le joueur avec le plus de points de victoire issus de ses cartes bleues l'emporte.

II Premier modèle conceptuel

II.1 UML

Un premier modèle d'UML a été réalisé en figure II.1. Il s'agit d'un modèle élémentaire d'UML qui pourra faire l'objet de modifications dann un avenir proche.

II.2 Description des classes, attributs et méthodes

II.2.a Classe Partie

Cette classe prend en charge différents paramètres et méthodes qui sont globales durant la partie.

Attributs

- `solde_militaire : signed int`
  Représente la position du pion militaire relativement au joueur 1. Si $|$`solde_militaire`$| \geqslant 9$, on proclame la fin de la partie par domination militaire.
- `nb_tour : int`
  Donne le tour actuel du jeu.
- `age : {1,2,3}`
  Donne l'âge actuel du jeu.
- `merveille_construite : int`
  Donne le nombre de merveilles construites par les joueurs.
- `Tour : {1,2}`
  Si `tour = 1`, c'est au joueur 1 de jouer. Sinon c'est au tour du joueur 2 de jouer.
- `disposition : Batiment tab[][]`
  Indique la position des bâtiments pour la pioche des cartes durant le jeu.

Méthodes

- `change_solde_militaire(Joueur, nbBoucliers) : void`
  Modifie le solde militaire et appelle `victoire_militaire` si $|$`solde_militaire`$| \geqslant 9$.
- `fin_age() : bool`
  Incrémente l'âge de `age` lorsque les conditions mettant un terme à l'âge 1 ou 2 sont remplies.
- `accessibilite(Batiment) : void`
  Change l'attribut `accessible` et `face_visible` dans la classe bâtiments si le tableau dans l'attribut `disposition` vérifie des conditions mathématiques sur les coefficients.
- `victoire_scientifique(Joueur) : void`
  Termine la partie lorsque les conditions de victoire scientifique sont réunies. Déclare un gagnant et un perdant.
- `victoire_militaire() : void`
  Termine la partie lorsque les conditions de victoire militaire sont réunies. Déclare un gagnant et un perdant.
- `change_solde_militaire(Joueur, nb_boucliers) : void`
  Modifie `solde_militaire` conformément aux paramètres renseignés.

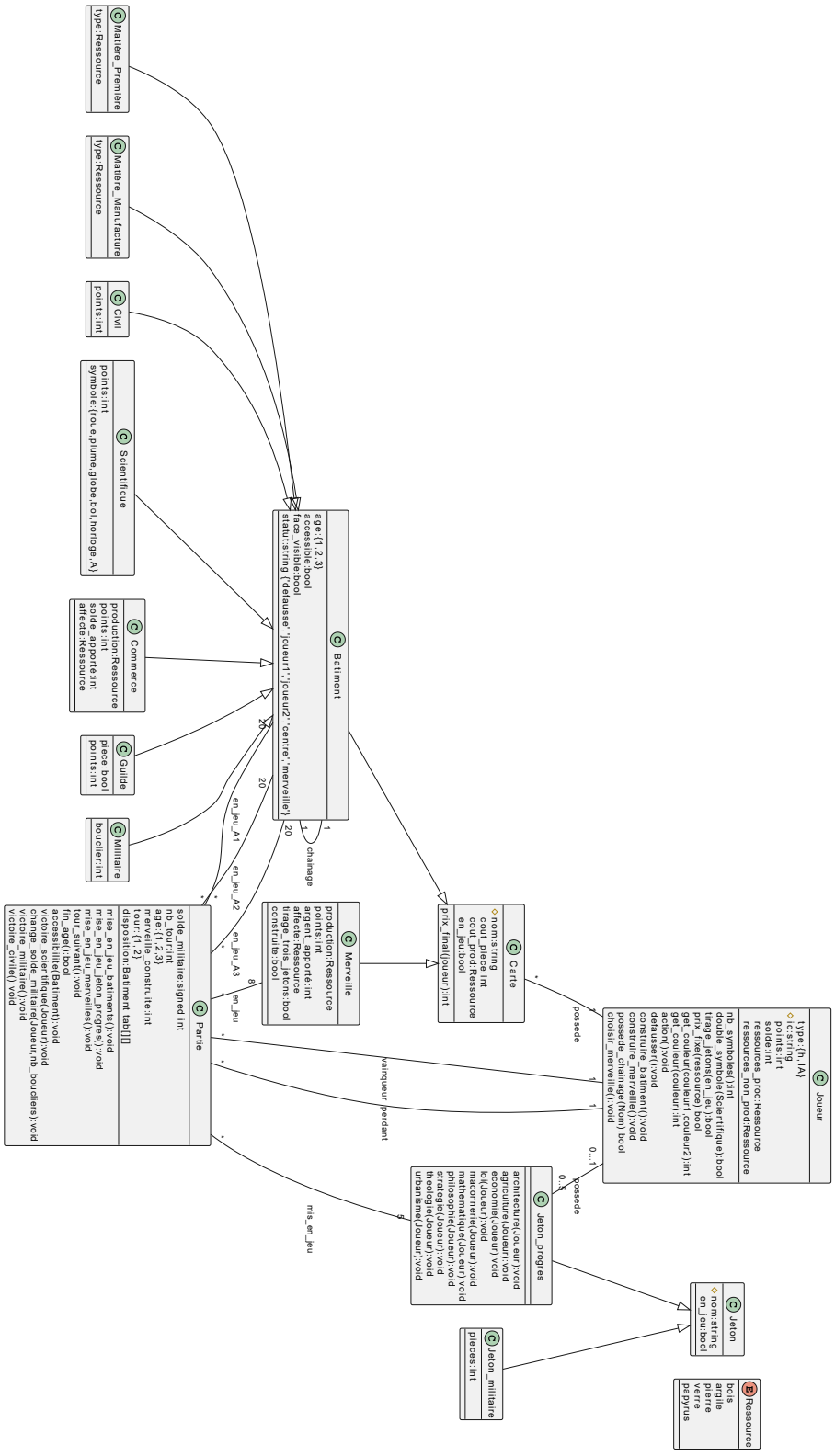

Figure II.1: Premier modèle d'UML

- `mise_en_jeu_batiments() : void`
  Désigne aléatoirement les bâtiments en jeu ou non.
- `mise_en_jeu_jeton_progres() : void`
  Désigne aléatoirement les jeton progrès en jeu ou non.
- `mise_en_jeu_merveilles() : void`
  Désigne aléatoirement les merveilles en jeu ou non.
- `victoire_civile() : void`
  Méthode appelée par la méthode `fin_age()` si c'est l'Âge 3 qui se termine. Permet de comparer les points de victoire de chaque joueur en faisant le tour de leurs attributs, et notamment en utilisant les pouvoirs des cartes Guilde en appelant les méthodes `get_couleur()` de la classe `Joueur`. Déclare un gagnant et un perdant.

II.2.b Classe Joueur

Attributs

- `type : {h,IA}`
  Permet de déterminer si le joueur est humain ou si c'est une IA.
- `id : string`
  Permet d'identifier les joueurs par leurs noms d'utilisateur. L'IA aura un id spécifique.
- `points : int`
  Comptabilise les points de victoire accumulés par le joueur.
- `solde : int`
  Comptabilise les pièces accumulées par le joueur.
- `ressources_prod : Ressource`
  Comptabilise les ressources produites directement par la cité du joueur.
- `ressources_non_prod : Ressource`
  Comptabilise les ressources produites indirectement (Bâtiments Commerce, Merveilles) par la cité du joueur, celles qui n'influent pas sur le prix du commerce.

Méthodes

- `nb_symboles() : int`
  Comptabilise le nombre de symboles scientifiques différents dans la cité en faisant le tour des cartes scientifiques possédées et du jeton loi éventuel, et met à jour l'affichage à l'écran. Appelle la méthode `victoire_scientifique()` de la classe Partie si la valeur retournée est 6.
- `double_symbole(Scientifique) : bool`
  Vérifie si le même symbole que celui du bâtiment entré en argument est déjà présent dans la cité. Permet de choisir un Jeton_Progrès si c'est le cas en appelant la méthode `tirage_jetons(en_jeu)`.
- `tirage_jetons(en_jeu) : bool`
  Permet au joueur de choisir un jeton Progrès et de l'ajouter à sa cité. Prend en argument un booléen indiquant si le jeton à choisir est en jeu ou non. Appelle la méthode de la classe Jeton_progrès correspondante au jeton choisi.
- `action() : void`
  Permet au joueur de choisir quelle action il souhaite entreprendre. Seules les bâtiments accessibles sont disponibles pour le choix. Change le statut de la carte choisie et appelle la méthode `accessibilité()` de la classe Partie. Appelle la méthode `prix_final()` de la classe Carte afin de seulement autoriser la construction du bâtiment et/ou de la merveille si le solde du joueur est assez élevé.
- `prix_fixe(ressource) : bool`
  Prend un type de ressource en argument, et retourne si oui ou non le prix est fixe pour le joueur concernant cette ressource (grâce à une carte commerce par exemple).

- `défausser() : void`
  Appelée par la méthode `action()` si le joueur choisit de défausser une carte pour des pièces. Met à jour le solde du joueur. Disponible pour toutes les cartes.

- `construire_batiment() : void`
  Appelée par la méthode `action()` si le joueur choisit de construire un bâtiment. Appelle les méthodes `nb_symboles()` et `double_symbole()` si le bâtiment choisi est scientifique. Appelle la méthode `change_solde_militaire()` si le bâtiment choisi est militaire. Met à jour l'attribut points si le bâtiment Civil est choisi. Met à jour l'attribut ressources si le bâtiment choisi est MatierePremiere ou ProduitManufacture ou Commerce qui produit des ressources. Appelle la méthode `Maconnerie()` de la classe Jeton_progrès si le joueur possède le jeton Maconnerie.

- `possede_chainage(Nom_Chaine) : bool`
  Vérifie si la cité du joueur possède une carte de chainage rentrée en argument. Appelée seulement si la carte choisie par le joueur est un bâtiment qui présente possède une association `chaînage` avec un autre bâtiment.

- `construire_merveille() : void`
  Appelée par la méthode `action()` si le joueur choisit de construire une merveille. Met à jour l'attribut construite de la merveille choisie, et l'attribut statut du bâtiment choisi pour la construire. Met à jour les attributs ressources, solde et points si nécessaire. Rappelle la méthode `action()` si la merveille permet de rejouer. Appelle la méthode `tirage_jetons(False)` si la merveille permet de tirer un jeton progrès. Appelle la méthode `Architecture()` de la classe Jeton_progrès si le joueur possède le jeton Architecture.

- `get_couleur(couleur1, couleur2) : int`
  Cette méthode permet d'utiliser les cartes Guilde à la fin de la partie en comptant le nombre de bâtiments d'un type rentrés en argument dans la cité du joueur. Elle est surchargée pour pouvoir compter le total d'un ou deux types.

- `choisir_merveille() : void`
  Permet au Joueur de choisir une merveille et de l'ajouter à sa cité.

II.2.c Classe Carte

Nous avons choisi de créer une classe Carte afin de pouvoir ajouter des classes qui en héritent si l'on souhaite ajouter d'autres types de cartes, comme par exemple les cartes des dieux pour l'extension du Panthéon. La classe Carte est reliée à la classe Joueur par l'association `possède`.

Attributs

- `nom : string`
  Toutes les cartes ont un nom unique qui les définit.

- `cout_piece : int`
  Coût de la carte en pièces.

- `cout_prod : Ressource`
  Coût de la carte en ressources.

- `en_jeu : bool`
  Attribut qui indique si la carte a été mise en jeu ou non.

Méthodes

- `prix_final(Joueur) : int`
  Renvoie le prix final d'une carte pour le joueur passé en argument, en fonction des ressources produites par ce joueur et des prix fixes dont il dispose éventuellement. Appelle la méthode `prix_fixe()` du joueur en question.

II.2.d Classe Bâtiment

Nous avons choisi de créer une classe abstraite Bâtiment, et de créer 7 classes différentes selon le type de bâtiment afin de pouvoir ajouter un nouveau bâtiment facilement par héritage si on le souhaite. Nous avons choisi de mettre trois relations différentes entre Partie et Bâtiment, pour indiquer qu'il y a 20 bâtiments en jeu pour chaque Âge. Les méthodes que nous implémenterons pour la classe Bâtiment en C++ serviront seulement à changer et mettre à jour les attributs de la classe, nous ne les détaillerons donc pas dans ce rapport. *Pour la classe bâtiment et ses classes filles, nous détaillerons seulement les attributs.*

Attributs

Bâtiment

- `age : {1,2,3}`
  Un bâtiment appartient à l'âge 1, 2 ou 3. On considérera que les cartes Guilde appartiennent à l'Âge 3.
- `face_visible : bool`
  Indique si le bâtiment disposé dans le jeu est face visible ou face cachée.
- `accessible : bool`
  Indique si le bâtiment est accessible ou non pour les joueurs. Un bâtiment face visible n'est pas forcément accessible, mais un bâtiment accessible est forcément face visible.
- `statut : string`
  {'defausse','joueur1','joueur2','centre','merveille'}
  Indique quel statut a la carte dans le jeu.

Matière Première

- `type : Ressource`
  Indique le type de ressources produites et leur quantité grâce à l'Énumération Ressource.

Matière Manufacturée

- `type : Ressource`
  Indique le type de ressources produites et leur quantité grâce à l'Énumération Ressource.

Civil

- `points : int`
  Indique le nombre de points de victoire apportés par le bâtiment.

Scientifique

- `points : int`
  Indique le nombre de points de victoire apportés par le bâtiment.
- `symbole : {roue, plume, globe, bol, horloge, A}`
  Indique le symbole scientifique que porte la carte.

Commerce

- `production : Ressource`
  Indique le type de ressources produites et leur quantité grâce à l'Énumération Ressource.
- `points : int`
  Indique le nombre de points de victoire apportés par le bâtiment.
- `solde_apporte : int`
  Indique les pièces apportées par la carte commerce.
- `affecte : Ressource`
  Indique le type de ressources dont le prix est fixé à 1 pièce grâce à l'Énumération Ressource.

Guilde

- `type : string`
  Indique si la carte Guilde doit comptabiliser un type de bâtiment, une merveille ou des pièces (marron et gris, merveille, jaune, bleu, rouge, vert, pièce).
- `piece : bool`
  Si une carte Guilde apporte des pièces ; elle n'en apporte qu'une par type de carte indiqué. Il est donc uniquement nécessaire de savoir si oui ou non elle apporte des pièces.
- `points : int`
  Points apportés par la carte Guilde (qu'on multipliera par le nombre de biens de type spécifié à la fin de la partie).

Militaire

- `bouclier : int`
  Nombre de boucliers apportés par la carte.

II.2.e Classe Merveille

Attributs

- `production: Ressource`
  Comptabilise les ressources produites par la merveille.
- `points: int`
  Points de victoire que rapporteront la merveille.
- `argent_apporte : int`
  Pièces de monnaie apportées par la merveille.
- `affecte: Ressource`
  Indique les ressources dont le prix est affecté par la merveille.
- `tirage_trois_jetons : bool`
  Indique si la merveille permet de tirer trois jetons.
- `construite: bool`
  Indique si la merveille est construite ou non.

II.2.f Classe Jeton

Nous avons choisi de créer une classe abstraire Jeton afin de pouvoir ajouter de nouveaux jetons par la suite si besoin dans une extension.

Attributs

Jeton

- `nom : string`
  Les jetons ont un nom, nous créerons un nom fictif pour les jetons militaires selon les pièces qu'ils saccagent et le côté du joueur duquel ils sont (1Jeton5, 1Jeton2, 2Jeton5, 2Jeton2).
- `en_jeu : bool`
  Certains jetons sont tirés au sort. Cet attribut indique donc s'ils ont été mis en jeu ou non.

Jeton Militaire

- `pieces : int`
  Montant des pièces que le jeton saccagera.

Méthodes *(Jetons Progrès)*

Cette classe est reliée à la classe Joueur par l'association `possède`. Comme les jetons progrès sont tous différents, *la classe n'a aucun attribut, seulement des méthodes.*

- `agriculture(Joueur) : void`
  Augmente le solde du Joueur de 6 pièces et augmente ses points de 4 unités.
- `architecture(Joueur) : void`
  Cette méthode sera appelée par la méthode `construire_merveille()` de la classe Joueur. Laisse le joueur choisir deux ressources gratuites pour construire la merveille et modifie la valeur renvoyée par la méthode `prix_final()` de la classe Carte.
- `economie(Joueur) : void`
  Le joueur récupère l'argent dépensé par son adversaire lorsqu'il achète des ressources.La méthode `prix_final()` nous donne le prix final d'une carte en fonction des ressources qu'il a potentiellement acheté, donc le prix des ressources est `prix_final() - cout_piece()`.Il suffit donc de récupérer cette valeur à chaque tour.
- `loi(Joueur) : void`
  Appelle la méthode `nb_symboles()` de la classe Joueur.
- `maconnerie(Joueur) : void`
  Cette méthode sera appelée par la méthode `construire_batiment()` de la classe Joueur. Laisse le joueur choisir deux ressources gratuites pour construire le bâtiment civil et modifie la valeur renvoyée par la méthode `prix_final()` de la classe Carte.
- `mathematiques(Joueur) : void`
  Apporte 3 points de victoire pour chaque Jeton Progrès en la possession du Joueur, en modifiant l'attribut points du joueur passé en argument.
- `philosophie(Joueur) : void`
  Apporte 7 points de victoire au Joueur passé en argument en modifiant son attribut points.
- `stratégie(Joueur) : void`
  A partir du moment que le joueur possède jeton n'importe qu'elle carte `Militaire` qu'il tirera se verra attribuer un bouclier de plus. Donc en modifiant sont attribut bouclier.
- `théologie(Joueur) : void`
  Exécute la méthode `joueur::rejouer()` sous condition que la merveille ne possède pas déjà l'effet "rejouer".
- `urbanisme(Joueur) : void`
  Incrémente de 6 l'attribut `solde` du Joueur passé en paramètre. De plus dans le tableau Carte[][] si le joueur possède une nouvelle carte, donc que Carte[][] augmente de 1 sans que le solde du joueur augmente alors il l'attribut `solde` du Joueur augmentera de 4.

III Bilan

III.1 Tâche accomplie

- Valentin : Conception de l'UML, rédaction du chapitre II.
- Chloe : Conception de l'UML, rédaction du chapitre II.
- Joshua : Conception de l'UML, mise sous forme plantuml.
- Nadji : Rédaction du chapitre I.

III.2 Répartition des tâches futures

- Joshua : Implémentation de la classe Carte et ses classes filles
- Chloe : Implémentation de la classe Joueur
- Valentin : Interface graphique
- Nadji : Implémentation de la classe Jeton et ses classes filles
- Groupe : Implémentation de la classe Partie

III.3 Synthèse

Pour ce premier rendu, nous avons donc explicité notre vision du jeu au moyen d'un diagramme UML, avec des classes expliquées dans le chapitre II. Néanmoins, il reste des éléments plus complexes, tels que les méthodes des jetons progrès qui affectent énormément le déroulement de la partie. De plus, il y a la classe "Merveille" qui, si elle est utilisée, doit utiliser une carte. Enfin, le fait que certaines cartes soient faces cachées changera certainement l'interface graphique.