Citadelle

Un projet réalisé dans le cadre du cours
Objet et développement d'applications
François Hallereau
Sébastien Vallée

Décembre 2014

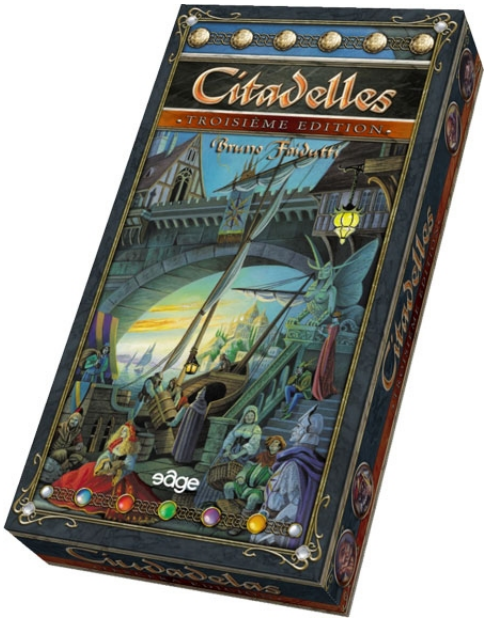


Table des matières

Table des figures

Introduction

Ce projet implémente le jeu de société Citadelle en C++, ce jeu de société a été créé en 2000 par Bruno Faidutti. Chaque joueur à pour objectif de bâtir une citadelle le plus rapidement possible mais surtout la plus riche possible.

L'originalité de ce jeu réside dans la possibilité de choisir un personnage à chaque tour de jeu, conférant ainsi des bonus divers et varié permettant de renverser le cours du jeu.

Dans ce rapport, nous décrirons les choix d'implémentations que nous avons fait ainsi que les différents pattern que nous avons utilisé.

1 Manuel d'utilisation

1.1 Déroulement du programme

Au lancement du programme tout est instancié. Conformément aux règles officielles, lors du premier tour, un joueur est choisi aléatoirement pour prendre possession de la couronne et ainsi débuter la partie. Chacun leur tour, les joueurs vont choisir un personnage qui va leur permettre d'acquérir une capacité spéciale qui sera effective uniquement durant ce tour. Une fois les personnages choisis, Le système va appeler les personnages chacun leur tour et permettant ainsi au joueur, qui s'est attribué le personnage, de commencer son tour.

Il peut ainsi :

— prendre 2 pièces d'or ou piocher 2 cartes puis en remettre une sous la pioche.

— poser une carte devant lui et donc l'inclure dans sa citadelle.

— activer sa capacité spéciale.

La partie se termine lorsqu'un des joueurs a devant lui le nombre de cartes requis mettant fin à celle-ci. Les joueurs font ensuite le décompte des points de leur citadelle. Le joueur ayant le plus de point remporte la partie.

1.2 Règle du jeu

Notre version du jeu se base sur la première édition du jeu elle ne comprend pas l'ajout des nouvelles cartes et notamment des derniers personnages.

Comme la version officielle, le jeu se joue de 3 à 7 joueurs.

Nous avons aussi laisser la possibilité de choisir l'argent disponible dans la banque ainsi que le nombre de quartiers requis mettant fin à la partie. Par défaut, ces valeurs sont conformes à la version officielle. C'est à dire, 30 pièces d'or dans la banque et 8 quartiers requis.

2 Les Patterns

2.1 Pattern Strategy

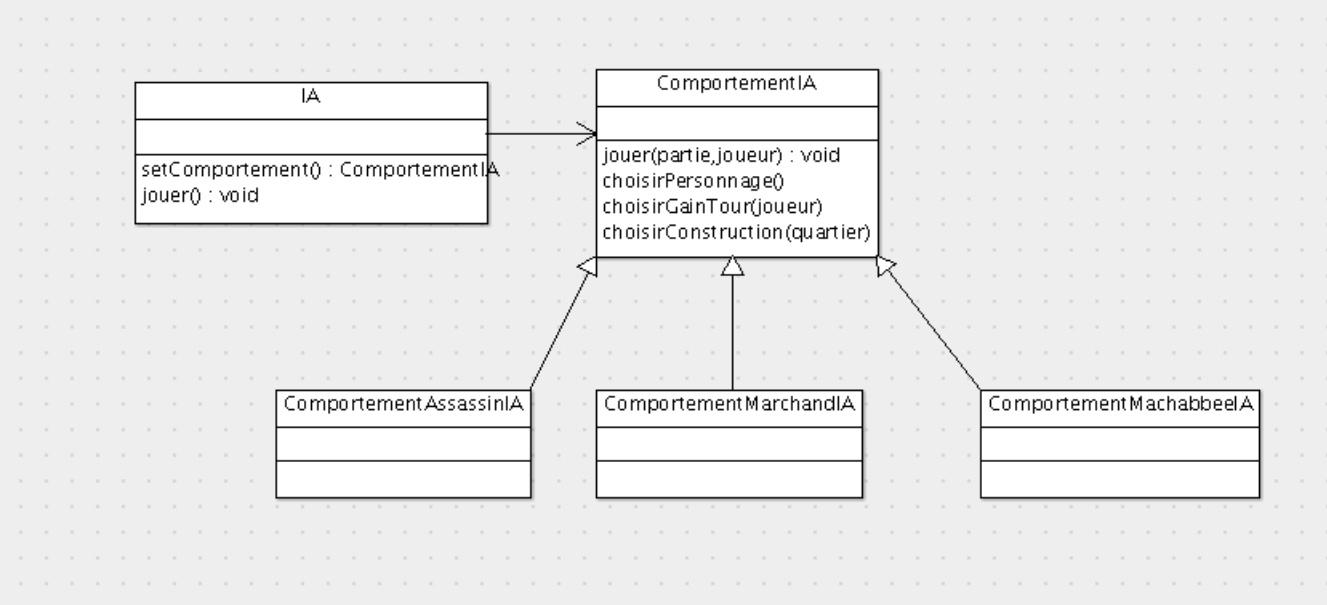


FIGURE 2.1 – Pattern Strategy
pour des raisons de clarté, seuls quelques comportements sont représentés

Le Pattern Strategy permet d'attribuer un comportement à un joueur en fonction de son personnage. Le comportement permettra de réaliser une liste d'action en fonction dudit personnage. Il est aussi utilisé lorsqu'un personnage se fait tuer par l'Assassin le forçant à passer son tour puis à s'annoncer à la fin du tour. L'avantage de ce pattern est qu'il permet la modification du comportement d'un joueur lors de l'exécution du programme.

2.2 Pattern Observer

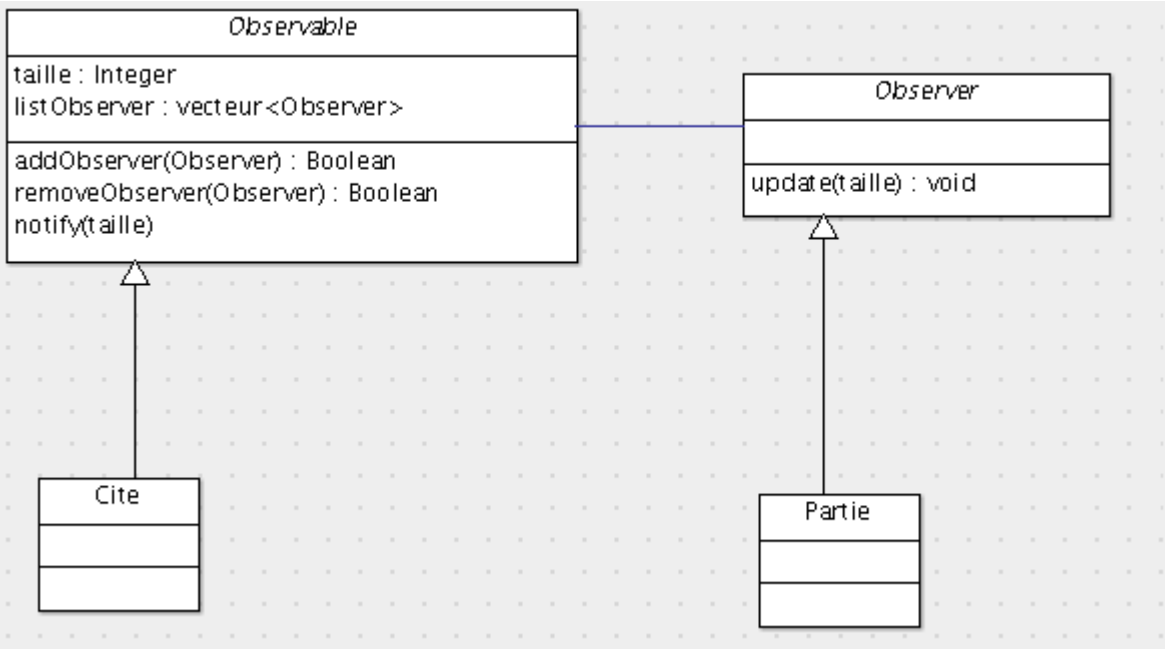


FIGURE 2.2 – Pattern Observer

Le pattern Observer est utilisé dans ce projet pour déterminer le moment où la partie doit se terminer. La classe *Cite* qui représente la citadelle des joueurs implémente la classe *Observable*. Elle gère l'ajout et la suppression de quartier en plus des méthodes hérités d'*Observable*.

La classe *Partie* implémente *Observer*. Elle gère tout le déroulement d'une partie, on peut ainsi la considérer comme étant le cerveau du programme.

De ce fait chaque citadelle envoie des mises à jour indiquant leur taille à la partie. Si l'une des tailles atteint la valeur maximale autorisée, la méthode update() fera appel à la méthode finduJeu().

2.3 Pattern Factory

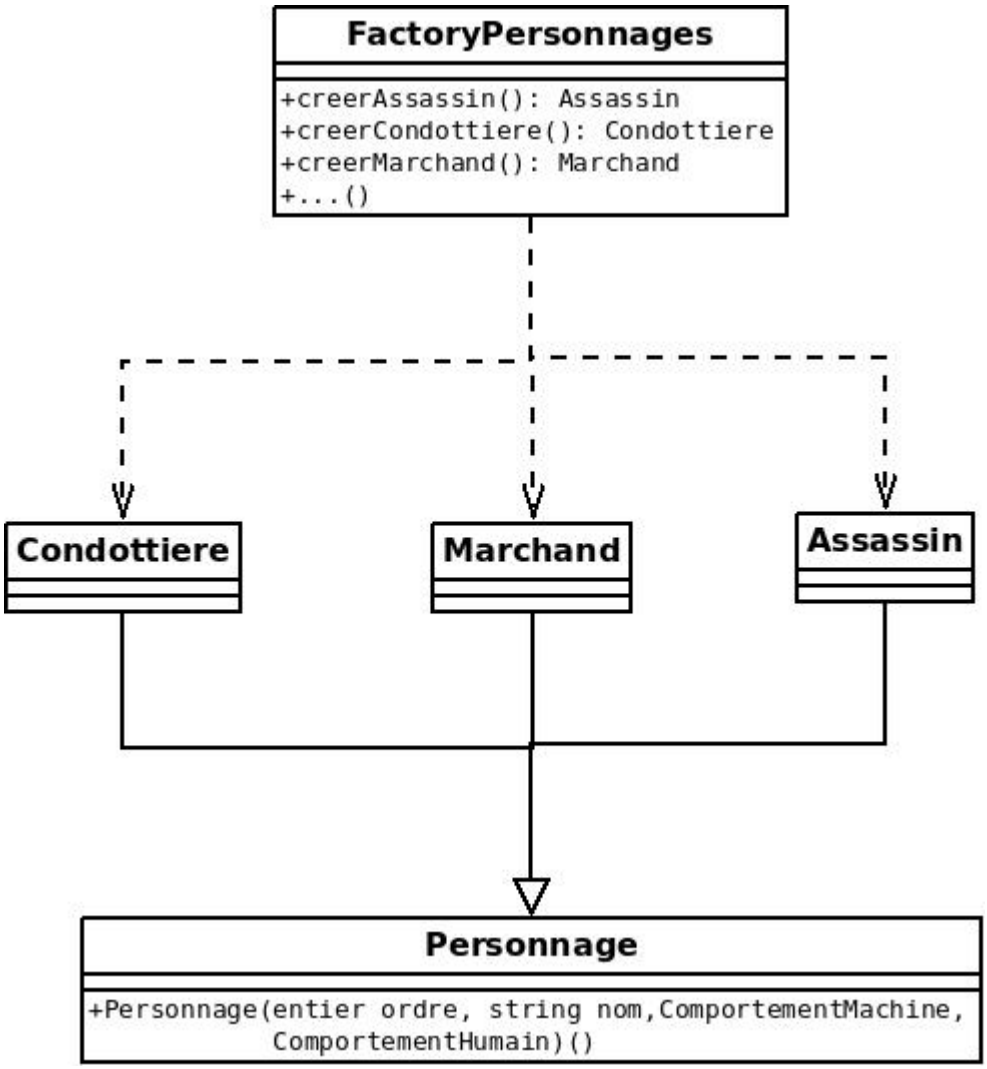


FIGURE 2.3 – Pattern Factory

Le pattern Factory est utilisé ici pour éviter au programme principal d'utiliser directement la création d'un personnage. Un Personnage est donc créé en appelant la méthode creerPersonnage() correspondante.La fonction se charge ensuite d'envoyer les bons paramètres au constructeur et de retourner cet objet. L'implémentation de ce patron de conception permet d'avoir et de modifier dans une seule classe le numéro, les différents comportements et le nom des personnages.Cela peut permettre par la suite de faire facilement des tests pour l'ajout de nouveau personnage et de changer rapidement son numéro.

3 Classes Importantes

3.1 Partie

La classe partie est une des classes,sinon la plus importante du programme. Elle permet de regrouper toutes les données de la partie(Asociations, pioche,limite de taille,..) et de permettre de savoir le moment où la partie doit être arrêtée, si les joueurs doivent choisir un personnage, si il y a un tour de jeu en cour,... Elle permet de regrouper et de lier la plupart des classes.

3.2 AssociationPersonnagesJoueurs

Cette classe permet de connaître le personnage d'un joueur, de savoir lequel doit commencer à choisir son personnage, le prochain joueur qui pourra jouer, etc. Cette classe permet à la classe partie d'organiser parfaitement le déroulement de chaque étape de la partie. Elle permet entre outre d'éviter la présence de deux joueurs avec les même pseudo, et empêche la présence de personnage ayant le même numéro d'ordre de passage.

3.3 Joueur

La classe abstraite Joueur est là pour permettre de regrouper les attributs essentielles à un joueur, qu'il soit un humain derrière la machine ou simplement la machine elle même. Cette classe ne peut être instanciée, un joueur étant forcément un humain ou une machine (comprendre ici un système jouant aléatoirement dans le cadre du débugage). Les informations essentielles sont par exemple le pseudo du joueur, son nombre de pièce d'or, le comportement qui lui est propre (en fonction de son personnage), sa cité, la partie à laquelle il appartient, les cartes qu'il possède dans sa main. Elle regroupe des actions que les joueurs peuvent faire tel que prendre des pièces d'or, piocher des cartes, compter le nombre de points de sa cité,etc.

3.4 Personnage

Cette classe abstraite permet de regrouper des attributs communs à chaque personnage. Ainsi, un personnage instanciable (Assassin,Marchand,...) aura forcément un ordre(numéro de tour), un nom et des comportements qui lui sont propres. ComportementAssassinMachine et comortementAssassinHumain pour le personnage assassin par exemple.

3.5 ComportementMachine

Cette classe est très importante car elle permet d'associer à un personnage un comportement de "machine" qui lui est propre, afin de donner la possibilité à un joueur Machine qui jouerait ce personnage durant le tour en cours d'utiliser les pouvoirs et les actions(pouvoirs) disponibles et d'agir en conséquences. elle ne possède que peut d'attributs car l'utilisation d'un comportement par un joueur se fait par les méthodes que le comportement contient. Un joueur ne modifie pas le comportement durant son utilisation.

4 Implémentation

4.1 Difficulté rencontrées

La grande difficulté rencontrée a été de compiler toutes les classes dans le bon ordre. En effet, beaucoup de classes sont liées entre elles de part les attributs qu'elles contiennent et les paramètres de ses méthodes. Inclure correctement et proprement des classes interdépendantes, tout en évitant des inclusions multiples à donc été un réelle casse-tête.

D'autre part, le langage c++ en lui même a posé des difficultés. Nous n'avons pas l'habitude de raisonner à base de pointeurs et de référence. Ainsi, une grosse partie du travail a été non pas de structurer le programme mais de comprendre et de corriger toutes les erreurs liées à ce type de données.

4.2 Extensions possibles

Le programme ici présenté possède des avantages pour implémenter des extensions possibles au jeu. Il est tout a fait possible d'ajouter des personnages supplémentaires, de créer assez de personnage pour permettre des partie à 8,9,..,15 joueurs avec les numéro d'ordre de personnage correspondant évidemment.

L'ajout de différentes IA (moyen,difficile,...) pour augmenter le niveau de difficulté est de même très facile : ajouter un classe IAMoyen par exemple et d'implémenter des méthodes de choix un peu moins rudimentaire.

U n développeur peut aussi créer une infinité de quartiers différents ainsi que des quartiers avec des pouvoirs, le tout en ne modifiant pratiquement aucunes classes.

Conclusion

Ce projet met en évidence l'intérêt des design patterns dans les projets de grandes envergures. Le projet devient alors plus facilement compréhensible, mais il est aussi plus aisé d'ajouter des fonctionnalités. Ce qui est un point non négligeable dans ce genre de projet. En effet, il n'est pas rare que les clients souhaitent ajouter de nouvelles fonctionnalités au cours du développement de l'application.

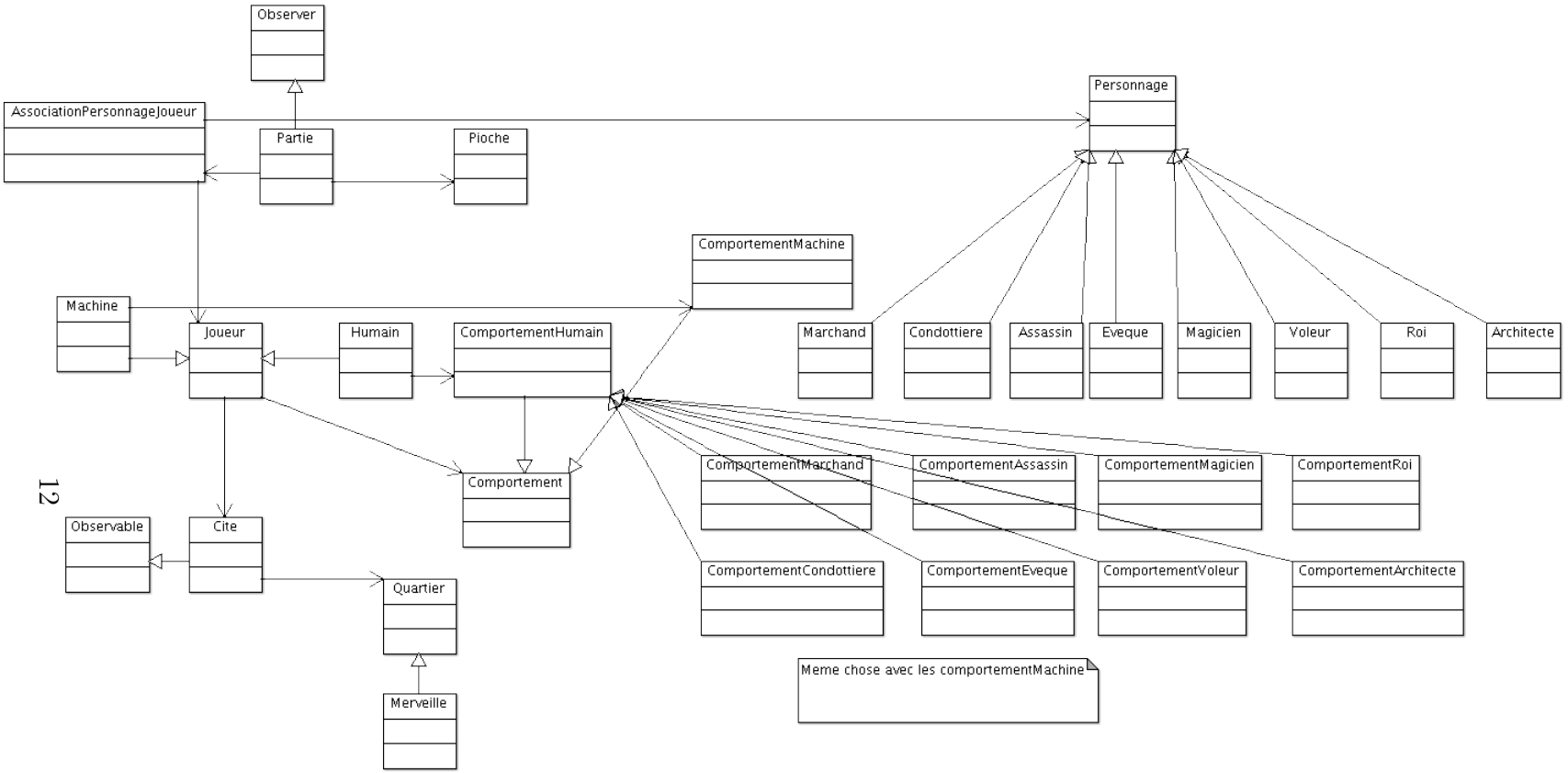
De plus, ce projet était un vrai défi dans le mesure où il n'y avait aucune limite, aucun sujet, la seule contrainte étant d'implémenter des patterns. Nous pouvions donc laisser exprimer notre imagination.

Cependant le manque de temps nous empêché d'implémenter toutes les fonctionnalités. Des cartes sont absentes et le manque d'une interface graphique rend l'expérience ludique moins intéressante. Aussi notre manque de connaissance sur les IA nous a contraint à en faire une que ne fait que des choix aléatoires.

Pour terminer, ce projet nous aura permis de développer en C++. Ce qui est une première pour un projet de cette envergure.

A Diagramme de classes

Ce diagramme décrit de manière succinte notre projet et met en évidence les différentes interactions entre nos classes.


Observer
AssociationPersonnageJoueur
Partie
Pioche
Personnage
ComportementMachine
Machine
Joueur
Humain
ComportementHumain
Marchand
Condottiere
Assassin
Eveque
Magicien
Voleur
Roi
Architecte
Comportement
ComportementMarchand
ComportementAssassin
ComportementMagicien
ComportementRoi
Observable
Cite
Quartier
ComportementCondottiere
ComportementEveque
ComportementVoleur
ComportementArchitecte
Merveille
Meme chose avec les comportementMachine