Jeu Civilisation

Rapport de conception

8 novembre 2012

Amandine Le Cahain
Adrien Brunelat

Introduction

Civilization est un jeu vidéo en tour par tour très connu des amateurs de jeux de stratégie. Pour ce projet d'Analyse, de Conception et de P.O.O, nous développerons une version minimale du jeu Civilization dans un univers un peu différent.

En effet, ici, les deux civilisations disponibles seront représentées par les deux départements EII et INFO. Les unités seront différenciées en 3 types à savoir « Directeur de Département », « Etudiant », « Enseignant » chacune ayant ses particularités.

Ce rapport présentera tous les documents réalisés lors de notre modélisation du jeu Civilization.

Contenu

Modélisation du jeu

1. Fonctionnalités illustrées par des cas d'utilisation

❖ Use Case 1: Units

Le cas d'utilisation représentant les **différentes actions des unités** est relativement basique puisqu'une unité ne peut que se déplacer, attaquer pour certaines et créer une ville pour d'autres. Une action indirecte de l'unité pourrait aussi être la défense lors d'une attaque mais nous avons choisi de ne pas le représenter dans le sens où l'unité ne choisit pas elle-même de défendre.

Une unité peut être soit un Directeur de département, un étudiant ou un enseignant représenté par un héritage.

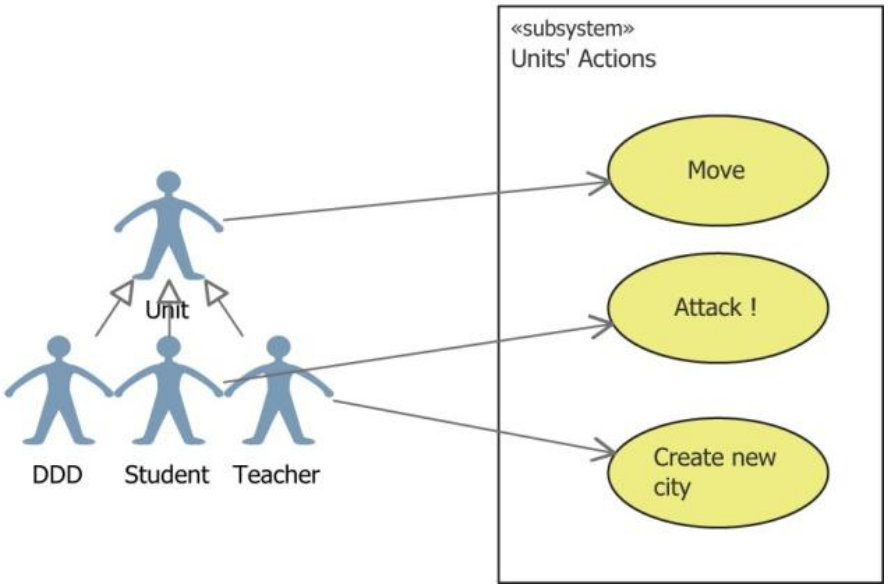

Figure 1 : Unit Use Case

❖ Use Case 2: Player

Un joueur possède de nombreuses **actions à réaliser** autant au niveau des menus du jeu que de la partie en elle-même. Un joueur peut donc créer une carte pour une partie et choisir sa taille. Il choisit sa civilisation puis il possède ensuite toutes les actions relatives à la partie. Les différents cas d'utilisation disponibles pour un joueur sont représentés sur la figure 2 ci-dessous.

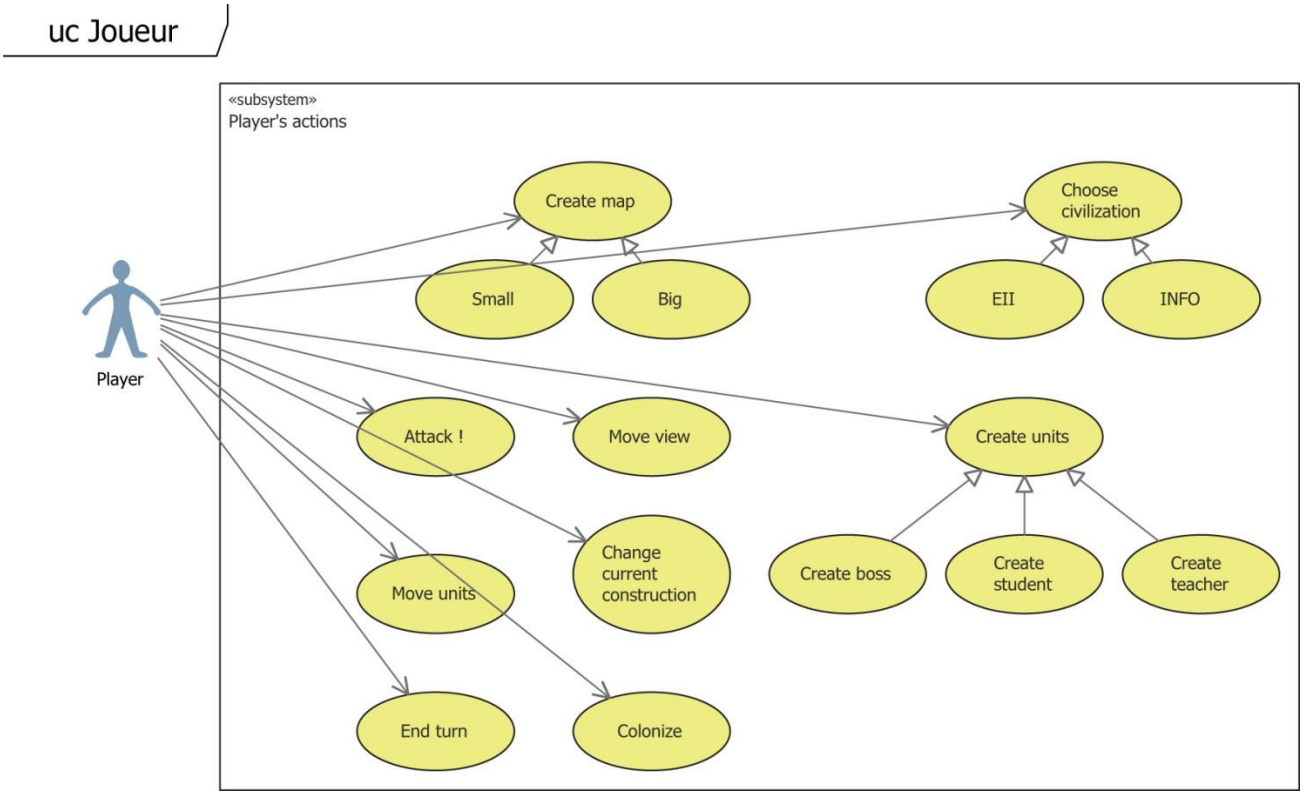

Figure 2 : Player Use Case

2. Modélisation des comportements du jeu

❖ Diagrammes d'interactions

Pour une meilleure lisibilité, les diagrammes présentés par la suite seront disponibles en annexe à la fin du rapport dans l'ordre où ils sont présentés.

Représentation de la création d'une partie

Afin de mettre en relation le monteur de partie avec les différents éléments qui lui sont liés, nous avons dessiné une séquence d'évènements pour la création d'une partie. La classe **« GameBuilder »** demande tout d'abord à la classe **« MapStrategy »** de créer une carte et de lui rendre. Ensuite, une liste de joueurs comprise entre 2 et 4 est créée et rendue au GameBuilder. Enfin, avec ces deux données, le GameBuilder peut demander la création d'une partie.

Représentation d'un tour de jeu

Dans un tour de jeu, le joueur peut :

- Effectuer les **actions relatives** à ses unités (déplacement, attaque, construction) ;
- Gérer les **productions de ses villes** (lancement et changement de production).

Le joueur peut aussi **passer le tour** de ses unités s'il ne désire faire aucune action.

Certains de nos choix de conception sont soulignés par ce diagramme, notamment :

- Chaque instance de la classe **« Case »** possède **sa propre liste d'unités**. Lorsqu'une unité meurt, elle est supprimée de la case sur laquelle elle se trouvait. Pour des raisons de praticité, l'unité possède aussi **deux attributs de coordonnées** afin de ne pas avoir à parcourir chaque liste de chaque case pour la retrouver ;
- Les unités ont chacune **un temps de production** sous forme d'un nombre de tours. En effet, pour des raisons de gameplay, lorsqu'une unité meurt, elle ne doit pas pouvoir être reproduite trop rapidement. Par exemple, le Directeur qui apporte un bonus considérable aux autres unités prendra plus de temps à être recréé.

Représentation d'une partie complète

La représentation d'une partie complète présente quelques simplifications pour plus de lisibilité. Nous avons représenté le **début de partie** où le joueur choisit sa civilisation. Celle-ci gère ensuite la création de ses unités de base et l'enseignant peut créer une ville.

Dans un second temps, nous avons représenté ce que le jeu devra tester à chaque début de tour. A chaque début de tour, le jeu doit se renseigner sur la **présence ou non d'un vainqueur** ; un joueur est gagnant s'il est le seul à posséder encore au moins une ville. Si le joueur testé n'est pas gagnant, le jeu regarde s'il est perdant c'est-à-dire, s'il ne possède aucune ville. Dans ce cas-là, le joueur est ajouté à la liste des perdants et devient **spectateur** de la partie. Si le joueur ne remplit aucun de ses deux conditions, le jeu démarre son tour.

❖ Diagrammes d'états-transitions d'un cycle de vie d'une unité

La figure 3 suivante présente la représentation d'un cycle de vie d'une unité suivant les tâches qu'elle aura à exécuter. Une unité peut être soit **morte ou vivante**. Dans sa vie, elle pourra passer par différents états **: en défense, en attaque, sans agression**.

Lors d'un combat, l'unité est engagée pendant un certain nombre de tours choisit aléatoirement entre 3 et le nombre de points de vie de l'unité ayant le plus de vie (située sur la même case) où s'additionne 2. Ceci explique les boucles sur les états de défense et d'attaque.

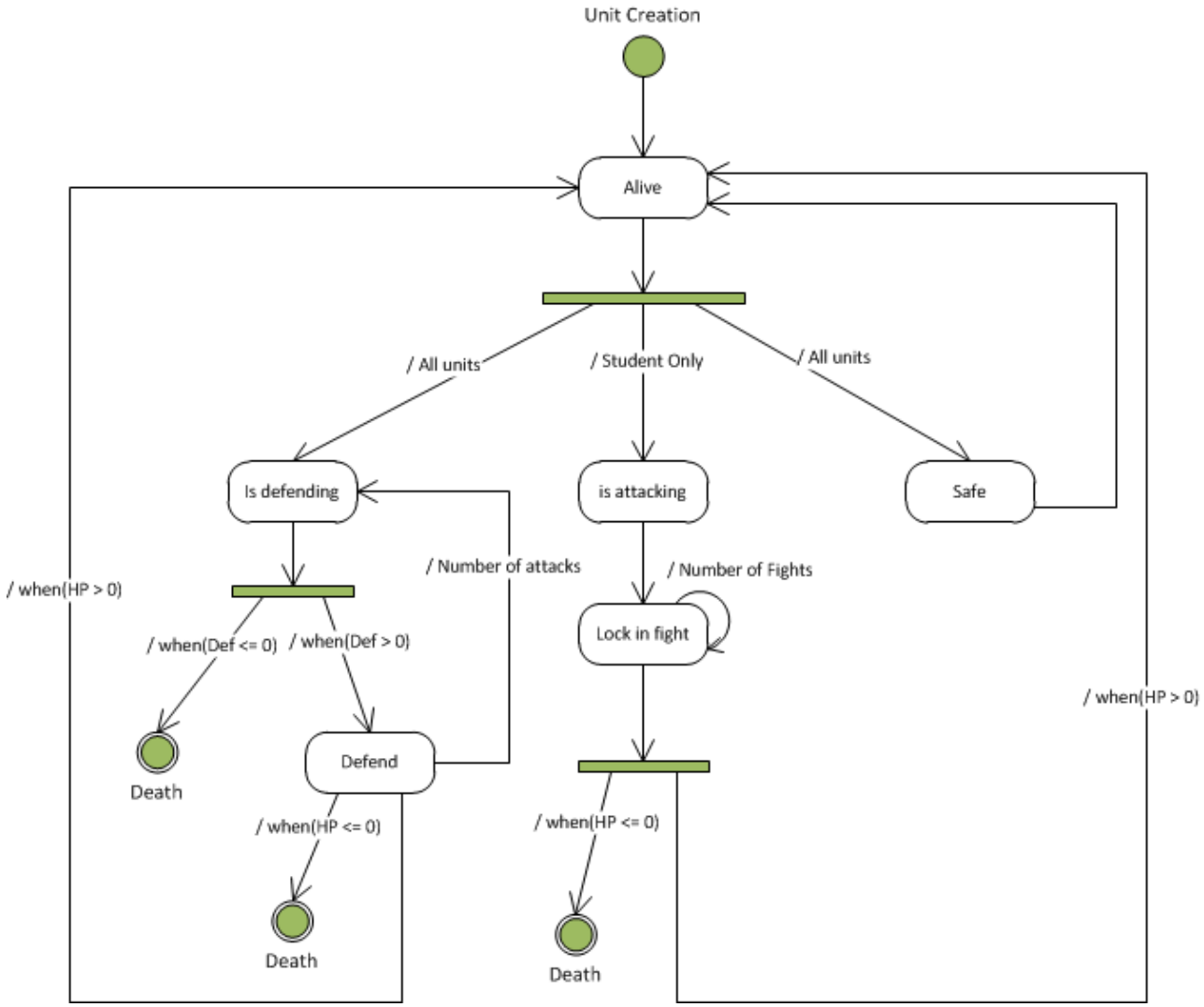

Figure 3 : Cycle de vie d'une unité

3. Modélisation des données et patrons de conception

❖ Diagramme regroupant les interfaces (API)

Fabrique abstraite : Création des unités

Nous avons utilisé le patron de conception Fabrique Abstraite permettant la fabrication d'objets concrets pour la gestion de la création des unités. Notre interface « **CivilizationFactory »** a donc pour rôle de créer les unités d'une civilisation. Elle donnera deux implémentations : **INFOFactory et EIIFactory** comme présenté sur les deux figures suivantes :

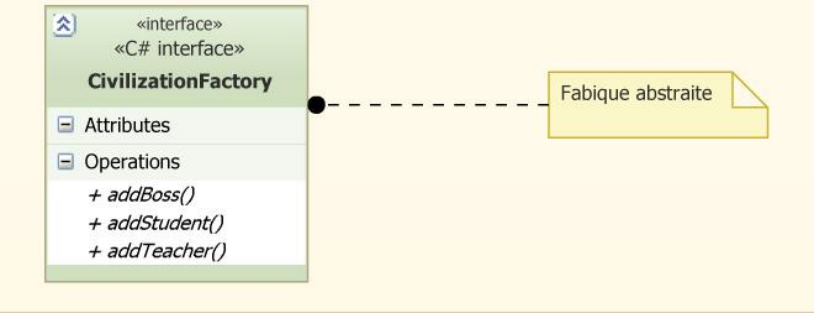

Figure 4 : Interface CivilizationFactory

Figure 5 : Implémentation CivilizationFactory

Monteur : Création de la partie

Le **monteur** est utilisé pour la création d'un objet complexe. GameBuilder va ici nous permettre de gérer la création d'une partie comme représenté sur la figure suivante:

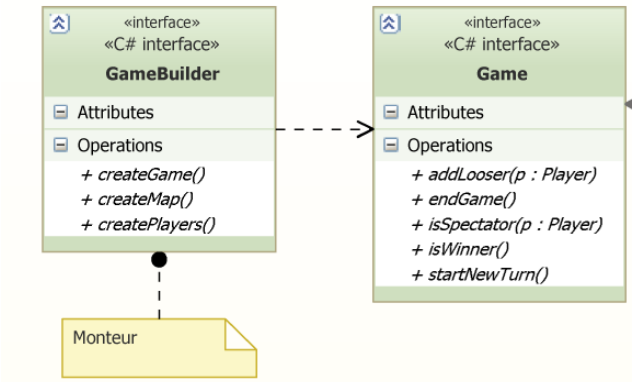

Figure 6 : Patron de conception Monteur

Figure 7 : Implémentation Monteur

Poids-mouche et décorateur : Création des cases et gestion de leurs ressources

La gestion des cases de la carte et notamment leur création sera attribuée à un poids-mouche pour faire apparaitre une nouvelle classe de construction des cases. Chaque case possède un nombre de ressources. L'attribution des ressources additionnelles aux différentes cases sera gérée par un décorateur comme le montre la figure 9:

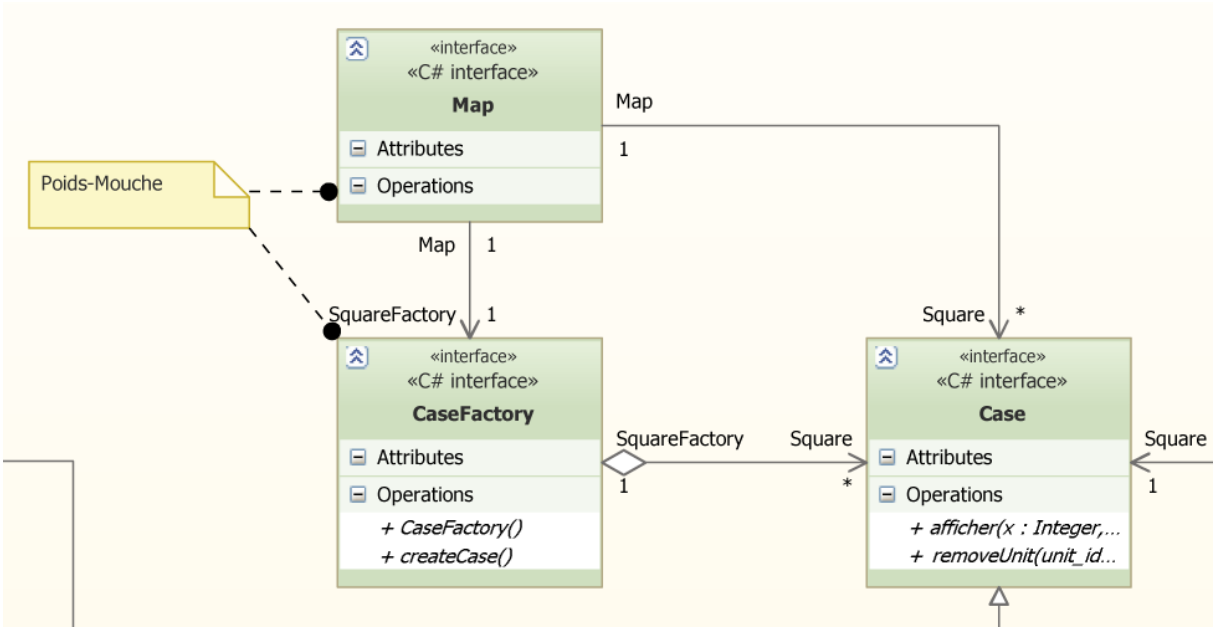

Figure 8 : Interfaces Poids-Mouche

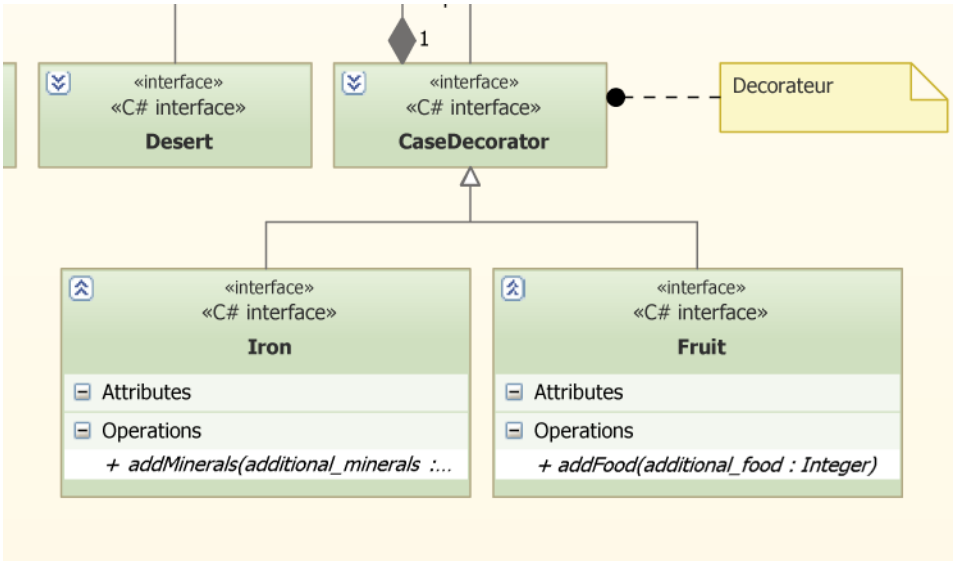

Figure 9 : Interfaces Decorateur

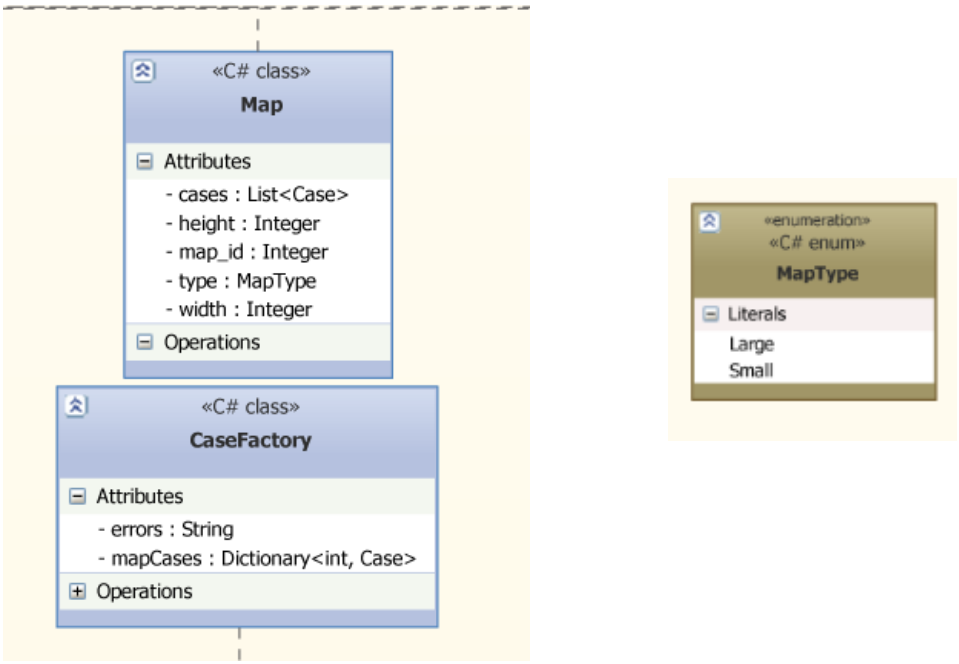

Figure 10 : Implémentation Poids-Mouche et énumération du type de carte

Nous avons adopté différents choix de conception fortement soulignés par ces figures:

- Une instance de la classe **« Map »** possède une liste de cases. Une case est facilement retrouvable parmi cette liste grâce à la hauteur et la largeur de la carte.
- La classe **« CaseFactory »** possède une hashmap contenant toutes les cases crées.
- Le type **MapType** provient d'une énumération {Large, Small}.

Stratégie : Création de la carte

La carte peut être créée de deux tailles différentes ; une petite de 25 x 25 cases ou une grande de 100 x 100 cases. Pour gérer la création de ses cartes et pour que le joueur puisse choisir la taille de la carte sur laquelle il veut jouer, le patron « Strategy » **sépare l'algorithme de création de carte de la carte elle-même** comme représenté sur le diagramme suivant :

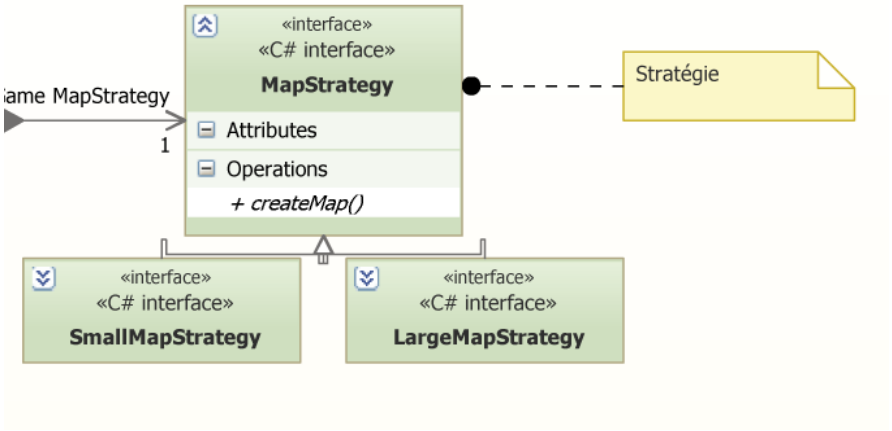

Figure 11: Patron Stratégie

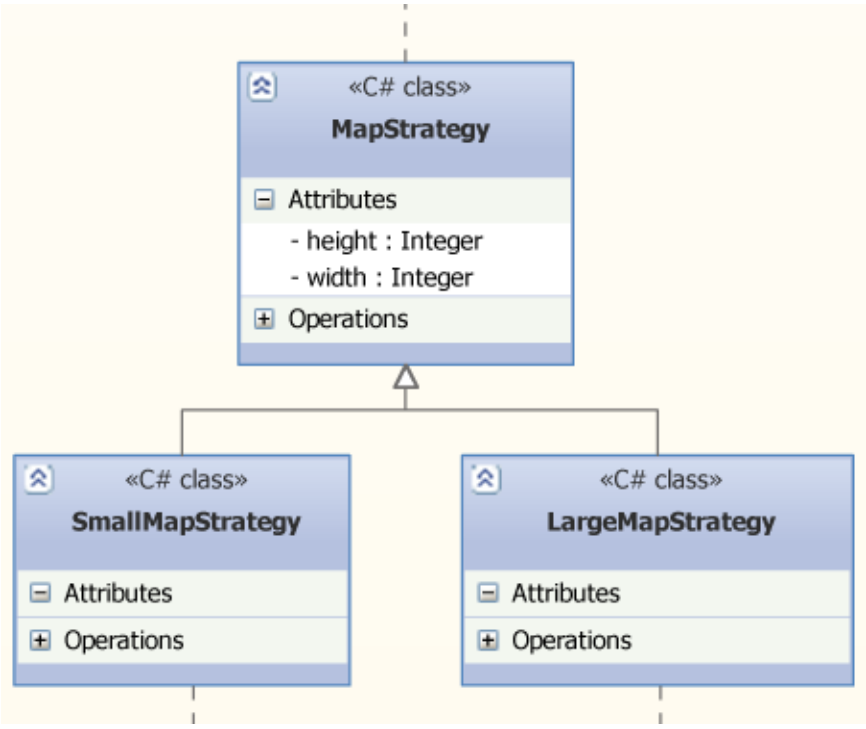

Figure 12 : Implémentation de la stratégie

Finalement, le diagramme de classes en annexe donne l'ensemble des interfaces qui seront nécessaires à l'implémentation de notre jeu.

❖ Diagramme des classes d'implémentation

S'ajoutant à notre diagramme regroupant les interfaces, nous avons créé un diagramme des classes implémentant ces interfaces. Il est encore assez difficile de déterminer l'ensemble des attributs de chaque classe ainsi que le réel besoin de celle-ci. Nous n'hésiterons pas pendant notre implémentation à retoucher certains des attributs et ajouter ou supprimer des classes qui nous semblerait à l'avenir inutiles.

Les principaux choix d'implémentations soulignés par ce diagramme de classe sont :

- **4 énumérations** :
  - **CivilizationType** donne le type de la civilisation que possède un joueur ;
  - **StatusType** donne le statut du joueur (actuellement en jeu ou spectateur de la partie) ;
  - **ProductionType** permet de lister les 3 productions des villes (Directeur, Enseignant, Etudiant) ;
  - **MapType** liste les deux types de carte disponible.
- Une instance de la classe **« Player »** possède une liste de villes afin qu'un joueur connaisse toujours la liste des villes qu'il possède. Par ailleurs, chaque ville possède un attribut joueur attitré.
- Une instance de la classe **« Game »** possède une liste de joueurs ainsi qu'une liste de perdants et un joueur gagnant. Ces deux listes ainsi que le statut de joueur gagnant permettront par la suite de gérer les parties à 4 joueurs plus facilement.

Les autres particularités de conception ont été énoncées tout au long du rapport.