Projet de MOCI

Rapport RPG

Mathieu Breit

TelecomNancy

Contents

Chapter 1

Rapport sur les Design Patterns

1.1 Singleton

1.1.1 Description

Le **Singleton** `GameConfiguration` centralise les paramètres globaux du jeu (difficulté, taille max d'une équipe). Il garantit qu'il n'existe qu'une seule instance de configuration partagée à travers l'application.

1.1.2 Diagramme

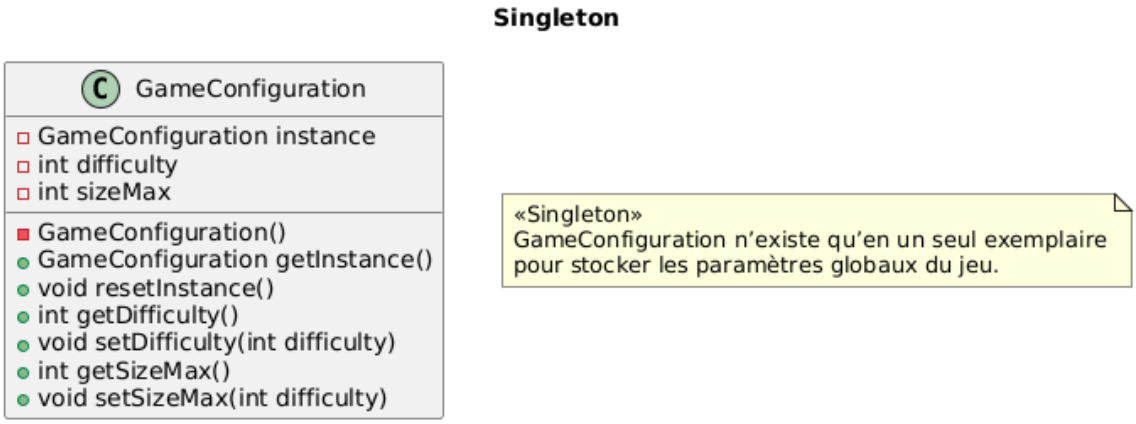

Figure 1.1: Diagramme du pattern Singleton

1.2 Factory Method

1.2.1 Description

La **Factory Method** permet de déléguer la création de personnages (`Warrior`, `Wizard`, `Healer`) à des classes spécialisées (`WarriorCreator`, `WizardCreator`, `HealerCreator`), sans exposer la logique d'instanciation au code client.

1.2.2 Diagramme

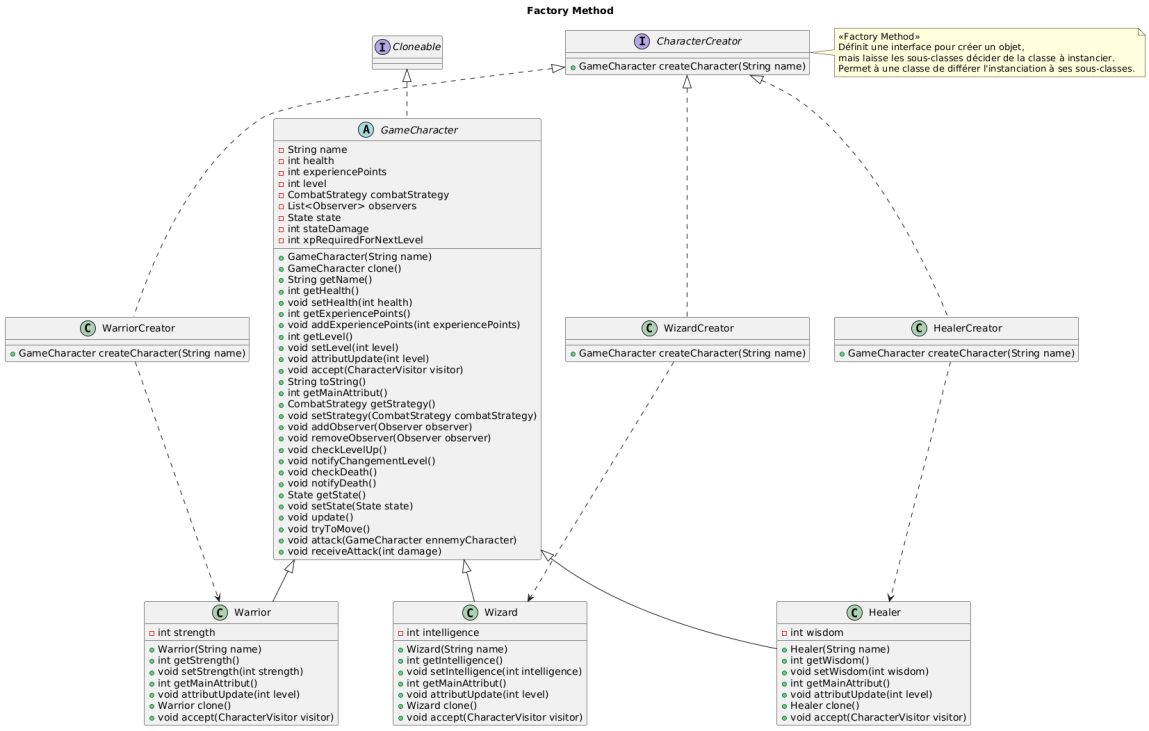

Figure 1.2: Diagramme du pattern Factory Method

1.3 Prototype

1.3.1 Description

Le **Prototype** `Team` implémente `Cloneable` pour faciliter la duplication profonde d'équipes et de leurs membres (`GameCharacter`). Cela permet de créer rapidement des équipes copiées d'un prototype initial.

1.3.2 Diagramme

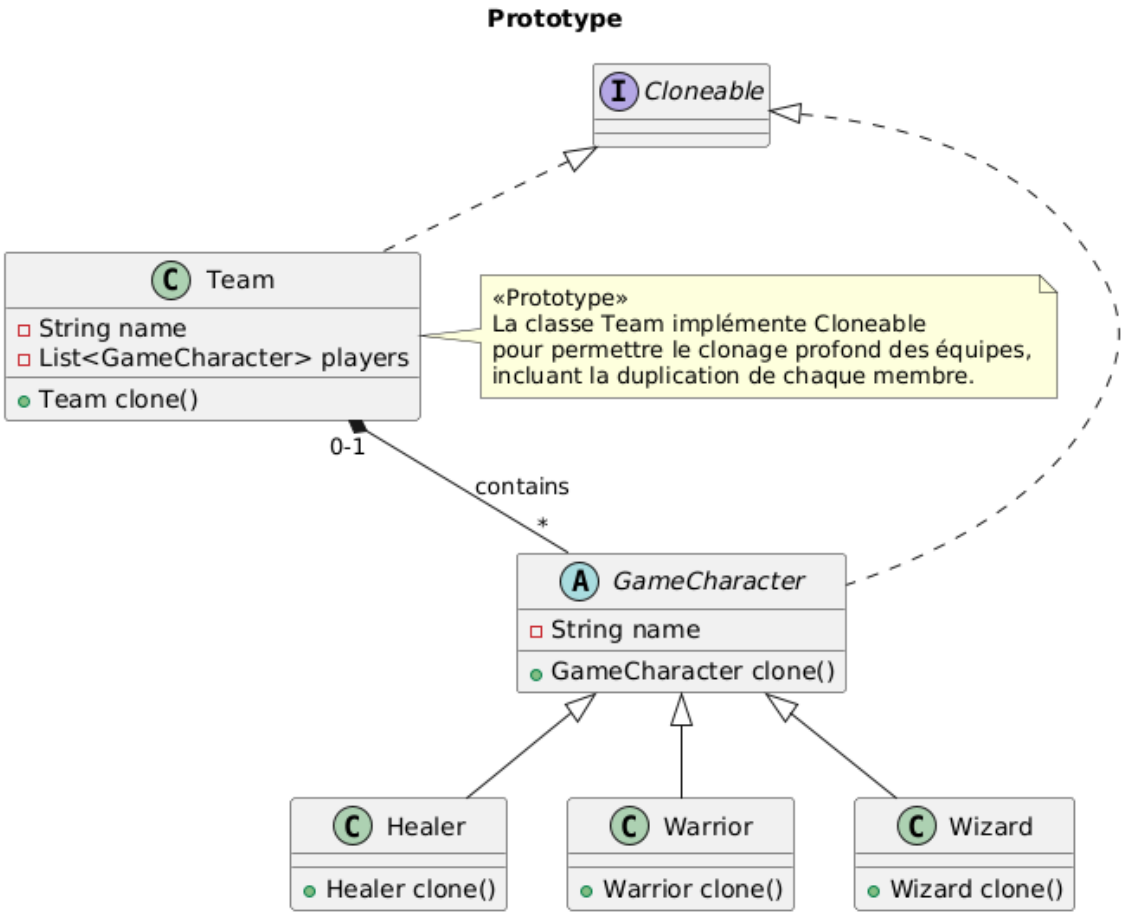

Figure 1.3: Diagramme du pattern Prototype

1.4 Builder

1.4.1 Description

Le **Builder** `TeamBuilder` fournit des méthodes pour construire progressivement une `Team` (ajout de personnages, définition du nom, etc.). Le `DirectorBuilder` orchestre la construction pour créer des équipes types (WizardTeam, etc.).

1.4.2 Diagramme

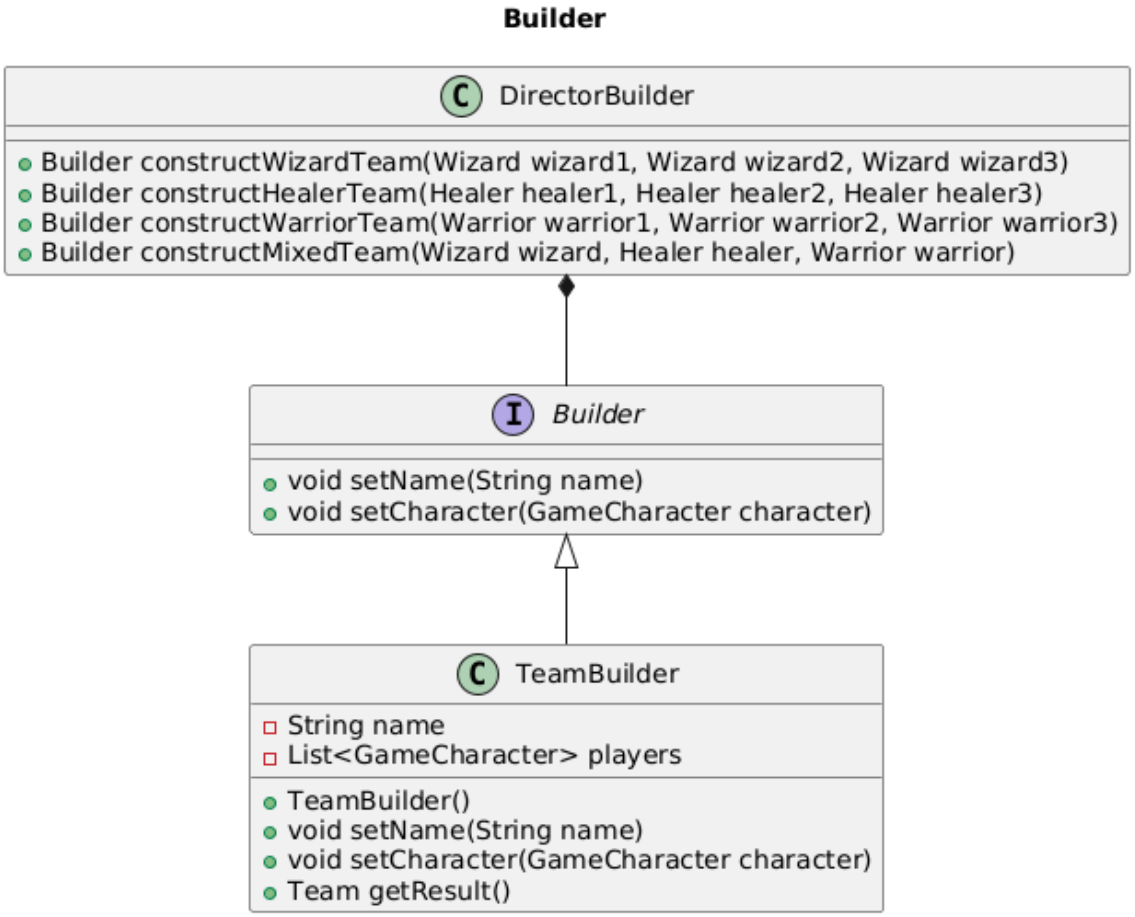

Figure 1.4: Diagramme du pattern Builder

1.5 Visitor

1.5.1 Description

Le **Visitor** introduit l'interface `CharacterVisitor` et des classes concrètes (`BuffVisitor`, `DamageVisitor`, `HealVisitor`) pour réaliser des opérations sur `Warrior`, `Wizard`, `Healer` sans devoir modifier leurs classes.

1.5.2 Diagramme

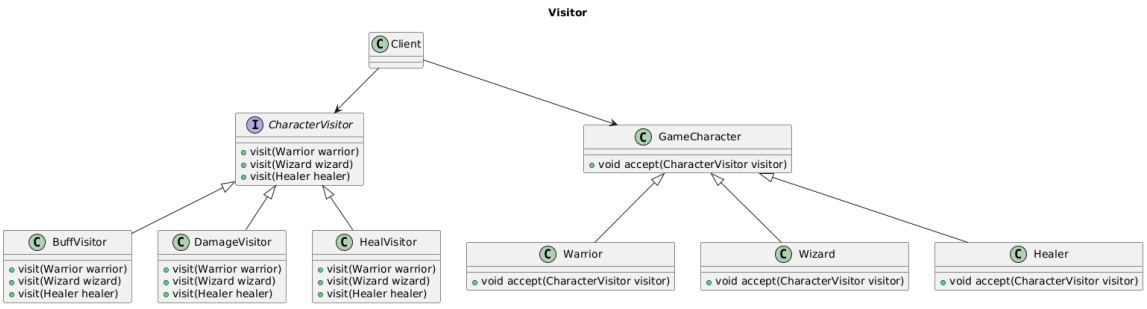

Figure 1.5: Diagramme du pattern Visitor

1.6 Strategy

1.6.1 Description

La **Strategy** `CombatStrategy` gère la façon dont un personnage calcule les dégâts qu'il inflige et qu'il reçoit (`Aggressive`, `Defensive`, `Neutral`). Cela permet de changer dynamiquement la stratégie de combat d'un personnage.

1.6.2 Diagramme

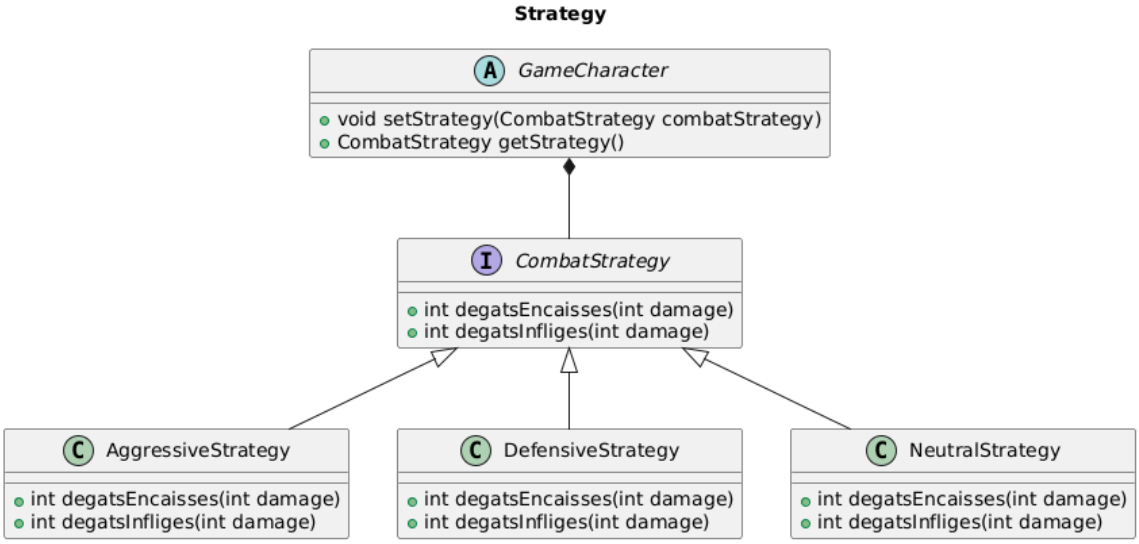

Figure 1.6: Diagramme du pattern Strategy

1.7 Observer

1.7.1 Description

L'**Observer** `CharacterObserver` (implémentations de `LevelUpObserver` et `DeathObserver`) permet à des observateurs de s'abonner à un `GameCharacter` et d'être notifiés lorsqu'il meurt ou monte de niveau en fonction de l'observateur. Lorsqu'un changement est effectué sur un personnage (augmentation de niveau ou mort), tous les observateurs abonnés à ce dernier sont informés.

1.7.2 Diagramme

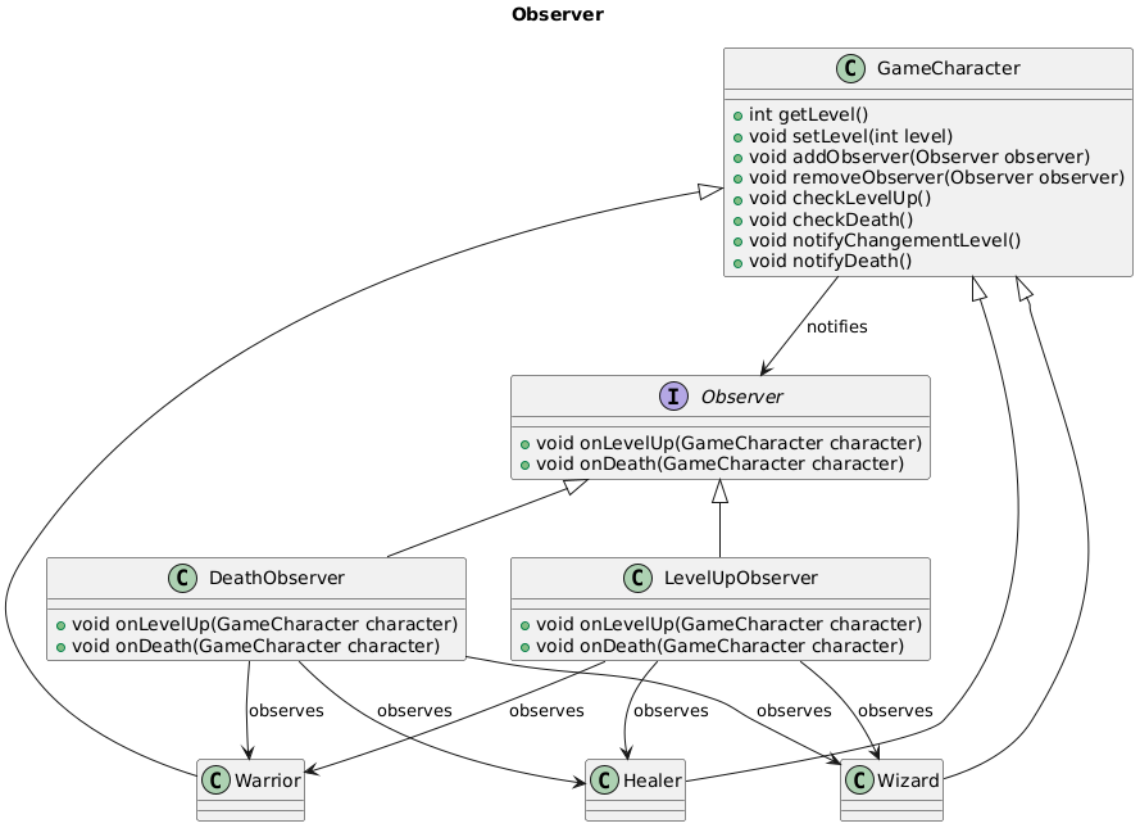

Figure 1.7: Diagramme du pattern Observer

1.8 Decorator

1.8.1 Description

Le **Decorator** enveloppe un `GameCharacter` pour modifier son comportement (diminution des dégâts reçus, invincibilité) sans multiplier les sous-classes (`ArmoredDecorator`, `InvincibleDecorator`). L'ajout et le retrait d'un ou de plusieurs décorateurs sur un personnage se fait dynamiquement. De plus, la structure du `GameCharacter` n'est pas modifiée.

1.8.2 Diagramme

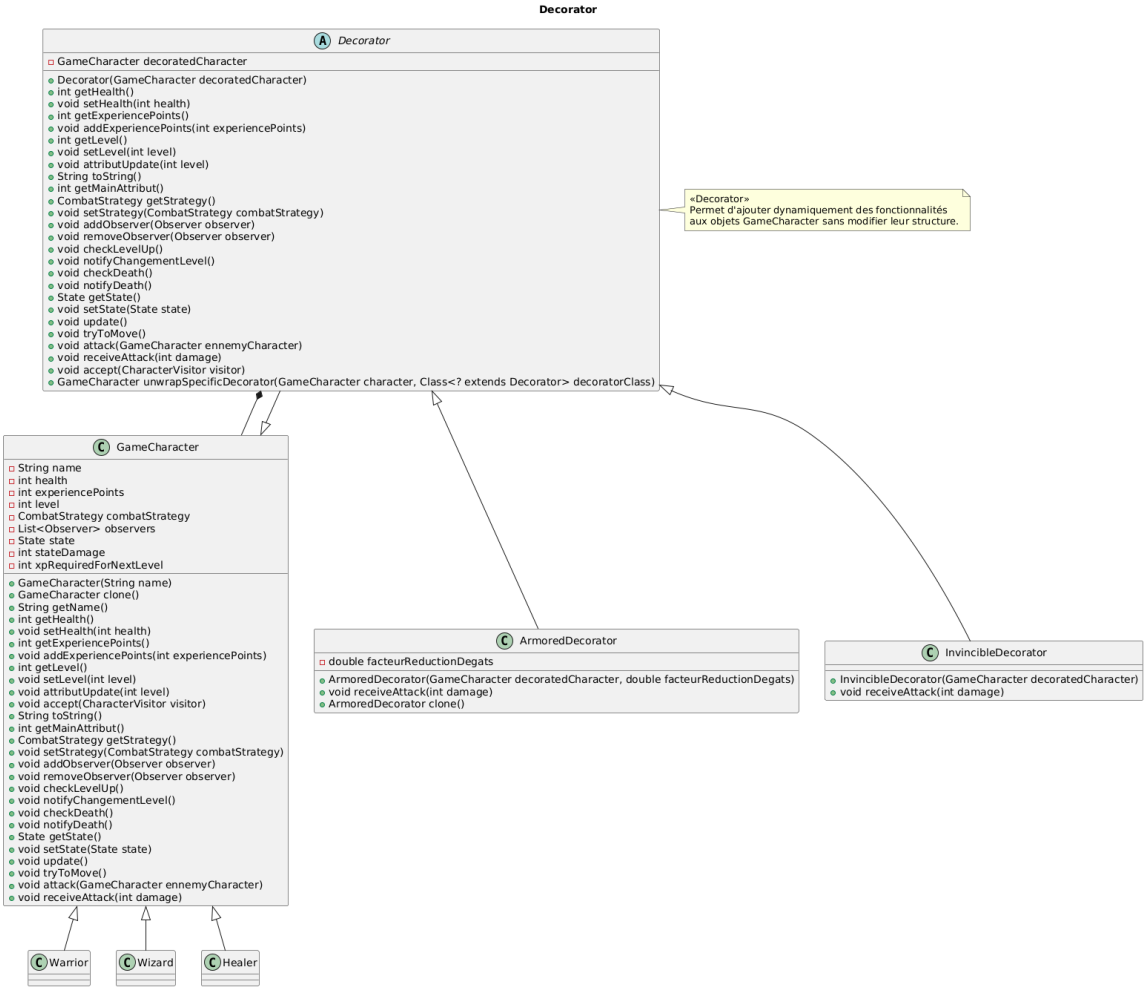

Figure 1.8: Diagramme du pattern Decorator

1.9 Command

1.9.1 Description

La **Command** encapsule chaque action du jeu (ajout/suppression d'équipe, attaque, heal, buff, etc.) dans un objet (`AddTeamCommand`, `RemoveTeamCommand`, ...) que l'`Invoker` (`GameInvoker`) exécute et annule au besoin, en faisant appel à la `Facade`.

1.9.2 Diagramme

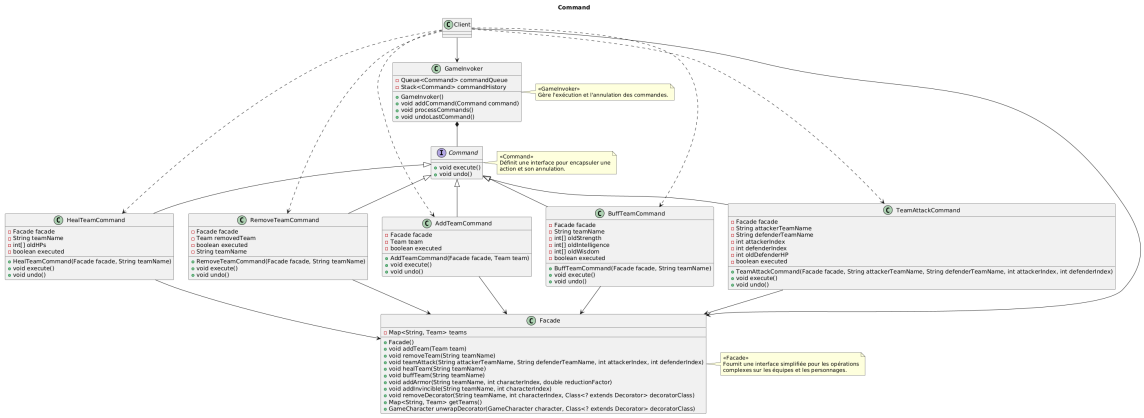

Figure 1.9: Diagramme du pattern Command

1.10 State

1.10.1 Description

Le **State** gère le comportement d'un personnage (`GameCharacter`) en fonction de son état (`NormalState`, `WoundedState`, `ScaredState`, `DeadState`). Chaque état redéfinit la mise à jour, le déplacement, l'attaque, etc. Par exemple, en fonction de son état, un personnage ne fera pas les mêmes dégâts lors d'une attaque, ou ne se déplacera pas à la même vitesse.

1.10.2 Diagramme

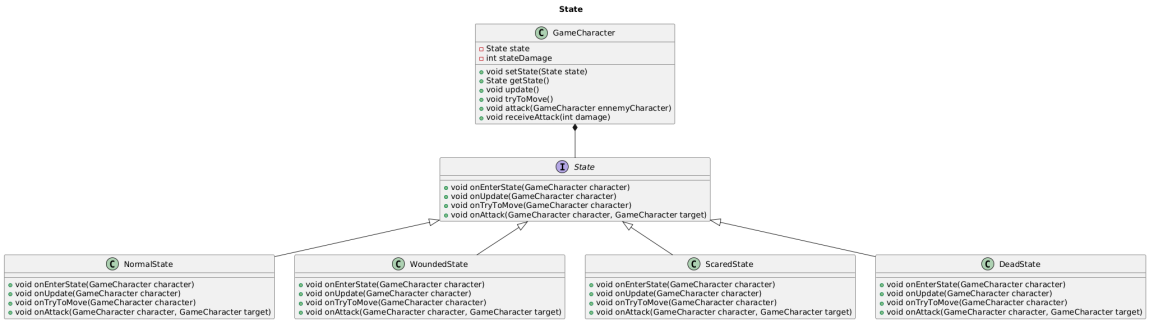

Figure 1.10: Diagramme du pattern State

1.11 Digramme complet

La mise en œuvre de ces divers patrons de conception a permis de résoudre de nombreux enjeux liés à l'implémentation du jeu RPG. Le diagramme ci-dessous réunit l'ensemble des classes et des patterns employés dans ce projet. Pour des raisons de lisibilité, seules les classes sont affichées, sans leurs attributs ni leurs méthodes.

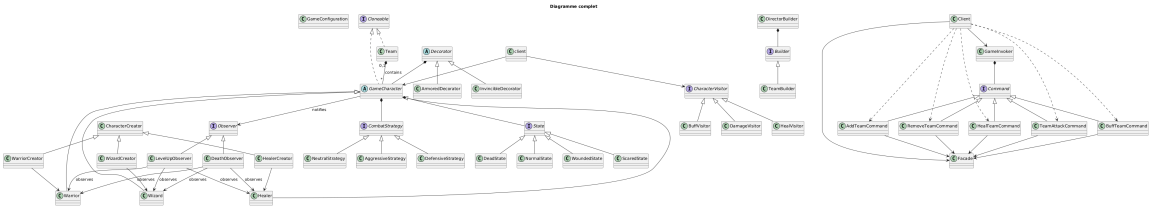

Figure 1.11: Diagramme complet