Projet Logiciel Transversal

Yoan AYROLES - Chloé PÉGUIN

Table des matières

1 Présentation générale

1.1 Archétype

L'objectif de ce projet est de réaliser un jeu semblable à **Fire Emblem 7** (soit **Fire Emblem : Rekka no Ken**), un jeu de rôle tactique au tour par tour de Intelligent Systems sorti en 2003 sur Game Boy Advance.

Figure 1 – Aperçu du jeu Fire Emblem 7 sur Game Boy Advance

1.2 Règles du jeu

Le joueur contrôle un groupe de personnages et affronte celui de son adversaire (contrôlé par l'ordinateur ou un autre joueur) dans une bataille tactique que le joueur remporte lorsqu'il atteint l'objectif défini en début de partie (capture d'une case, survie un certain nombre de tours, élimination d'un boss ou encore élimination totale des forces adverses).

Le déroulement d'un tour est le suivant : le joueur commence par déplacer une unité et lui donne des ordres (attendre, utiliser un objet ou encore attaquer). Une fois qu'il a fait cela pour chacune de ses unités, c'est au tour de l'adversaire de faire de même. Si toutes les unités d'un joueur sont tuées, il a perdu.

Les équipes de personnages sont constituées d'unités ayant des statistiques personnelles pour le combat (points de vie, attaque, défense, esquive…). Ces statistiques changent en fonction de la classe de l'unité (chevalier, archer, guerrier et brigand). À chaque classe correspond une arme précise (par exemple un brigand se bat à la hache, alors qu'un chevalier se bat à la lance). Une particularité des combats est le « triangle des armes » : les armes sont plus ou moins efficaces les unes contre les autres (l'épée est efficace contre la hache, qui est efficace contre la lance, elle-même efficace contre l'épée).

Chaque niveau se déroule sur une carte constituée de cases carrées sur lesquelles se déplacent les unités de chaque camp. Chaque case possède des caractéristiques qui influencent les combats en donnant des avantages aux combattants (par exemple, une unité sur une case « forêt » a plus de chance d'esquiver une attaque).

1.3 Ressources

Tout d'abord, nous utiliserons les textures de 16 pixels par 16 pixels pour la création de cartes du jeu original :

Figure 2 – Textures pour les cartes du jeu

Nous avons également les textures pour le lettrage, les jauges et les icônes du jeu :

Figure 3 – Textures pour le lettrage, les jauges et les icônes du jeu

Enfin, il y a les textures pour les personnages, de taille 16 pixels par 16 :

Figure 4 – Textures pour les personnages

2 Description et conception des états

2.1 Description des états

Un état de jeu est formé par un ensemble d'éléments fixes (les différents terrains qui composent la carte de jeu) et un ensemble d'éléments mobiles (les personnages). Tous les éléments possèdent les propriétés suivantes :

- Un nom
- Des coordonnées (x,y) dans la grille
- Des statistiques propres (PV, attaque, défense, esquive, critique)
- Un code de tuile

2.1.1 États éléments fixes

La carte de jeu est formée par une grille d'éléments de type Terrain. C'est une grille de taille 25 cases par 25 cases. Il existe différents types de terrains :

Terrains non praticables : ils sont inaccessibles aux éléments mobiles et n'ont donc aucune influence sur eux. Les terrains non praticables sont listés ci-dessous :

- Mur
- Rivière
- Montagne
- Falaise

Terrains praticables : ils sont accessibles aux éléments mobiles, ce sont les cases sur lesquelles les unités peuvent se déplacer. Chaque type de terrain praticable va influencer les statistiques des personnages en jeu. En effet les statistiques du terrain sur lequel se trouve un personnage s'ajoutent aux statistiques propres du personnage. On considère les types de terrains praticables suivants :

- Les plaines : elles ne changent pas les statistiques des personnages
- Les forêts : elles augmentent l'esquive des personnages situés dessus
- Les collines : elles augmentent la défense des personnages situé dessus
- Les ponts : ils ne changent pas les statistiques du personnage situé dessus mais permettent de traverser les rivières
- Les forteresses : à chaque tour elles redonnent plusieurs points de vie au personnage situé dessus sauf s'il a déjà tous ses points de vie.
- Les maisons : à chaque tour elles redonnent un point de vie au personnage situé dessus sauf s'il a déjà tous ses points de vie.

2.1.2 États éléments mobiles

Ces éléments sont dirigés par les joueurs. Ils pourront effectuer des actions (se déplacer, attaquer, attendre…) mais ces actions ne seront implémentées que plus tard dans le moteur du jeu. Les personnages possèdent une arme (identifiée par un code et un nom), un camp, un champ de mouvement, un champ d'attaque, un statut et un typeID.

Le statut représente l'état dans lequel se trouve un personnage à un instant donné :

- Sélectionné : dans le cas où le personnage a été sélectionné par le joueur pour effectuer une action.
- Disponible : dans le cas où le personnage peut-être sélectionné et n'a pas encore agi.
- Attente : dans le cas où le personnage ne peut-être sélectionné car :
  - c'est le tour du joueur adverse
  - le personnage a déjà effectué toutes les actions disponibles pour ce tour
- Mort : dans le cas où le nombre de points de vie du personnage vaut 0.

Le typeID représente le type auquel appartient un personnage :

- Archer
- Brigand
- Chevalier
- Guerrier

Les différents types de personnages sont différenciés par leur arme, leur champ de déplacement et d'attaque et leurs statistiques propres.

2.1.3 État général

A l'ensemble des éléments statiques et mobiles nous rajoutons la propriété suivante :

- tour : le nombre de tours qui ont été joués
- fin : un booléen qui indique si la partie est terminée

2.2 Conception Logicielle

Le diagramme des classes pour les états est présenté en Figure 5, dont nous pouvons mettre en évidence les groupes de classes suivants :

Classe Etat : Classe qui permet d'accéder à toutes les données qui forment un état. Elle contient un tableau à deux dimensions d'éléments de type Terrain formant la grille de jeu et un tableau à une dimension d'éléments de type Personnage se situant sur la grille.

Classes Element : Toutes les classes filles issues de la classe mère Element (beige) permettent de représenter les différentes catégories d'éléments (mobiles et statiques).

Classe Position : Classe qui décrit la position de chaque élément sur la grille de jeu.

Classe Statistiques : Classe qui regroupe les statistiques de chaque élément.

Classe Correspondances : Classe qui regroupe des sets de doublets composés d'un code de tuile et d'un ID d'élément.

Classes Observers : La conception de ces classes suit le Design Pattern Observer. Le but de la classe Observable dont hérite Etat est d'enregistrer des observateurs puis de les notifier à chaque changement d'état. Par exemple la classe StateLayer (render.dia) est un observateur et met à jour les textures/sprites à chaque action d'un personnage.

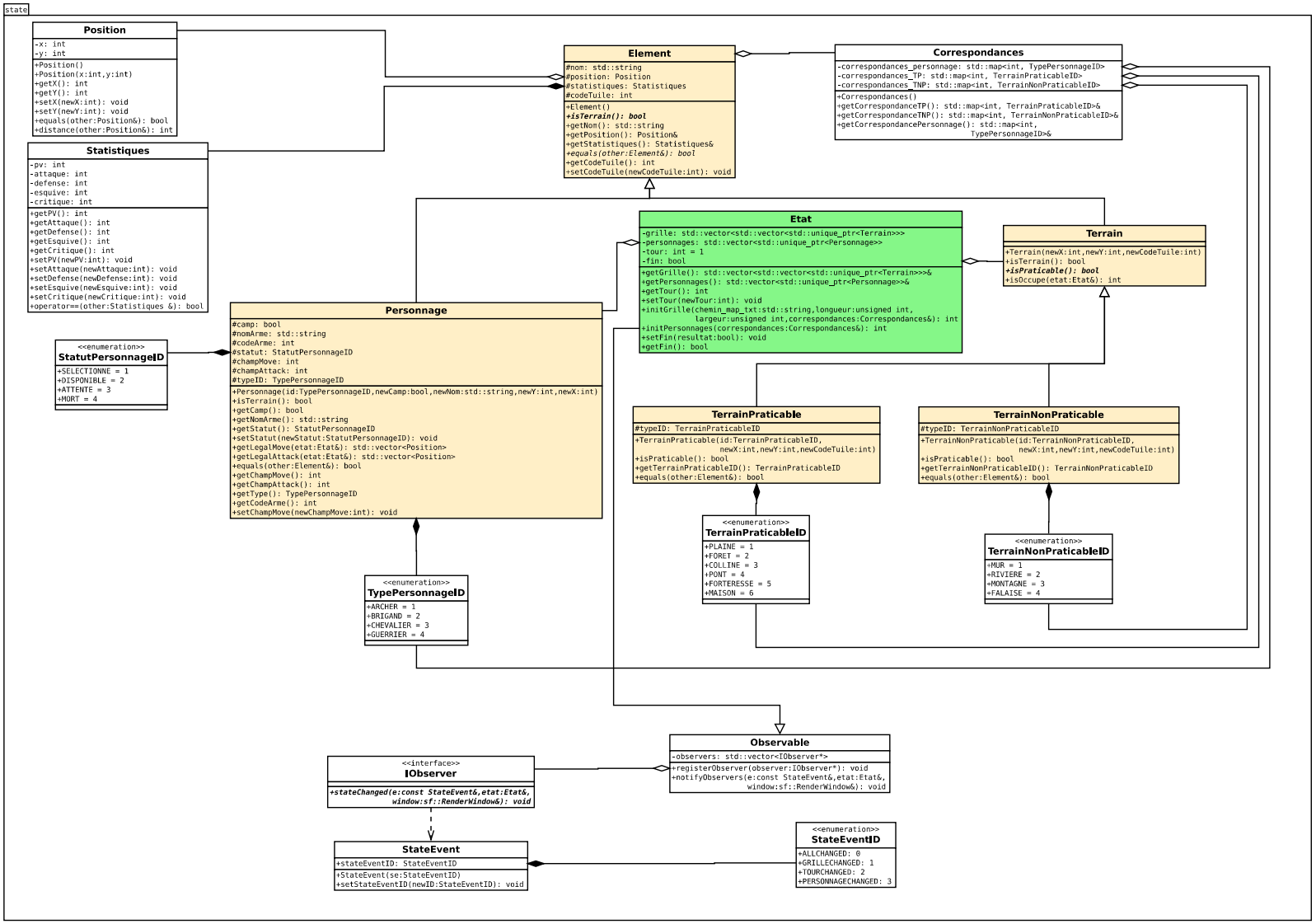

Figure 5 – Diagramme des classes d'état

3 Rendu : Stratégie et Conception

3.1 Stratégie de rendu d'un état

Pour le rendu d'un état, nous avons fait le choix d'un rendu par tuile à l'aide de la bibliothèque SFML.

Nous divisons la scène à afficher en trois surfaces : une surface contenant les éléments statiques Terrain, une surface pour les éléments mobiles Personnage et une surface pour les informations de jeu (statistiques et nom d'un personnage, tour de jeu…).

La grille de Terrain est créée à partir du fichier texte « ***map1.txt*** » composé de 25x25 codes de tuiles. La classe Correspondances (dans le package state) possède un tableau de correspondance pour les TerrainPraticables et un autre pour les TerrainsNonPraticables. Ces std::map associent à chaque code de tuile un ID de terrain spécifique. Par exemple, la première tuile du tileset pour la grille porte le numéro 0 et représente une MAISON, et cette association est donc répertoriée dans l'attribut correspondanceTP de la classe Correspondances. Pour modifier la grille il suffit donc de modifier les entiers dans le fichier *« **map1.txt** »* (qui se trouve dans res/)**.**

La méthode initGrille de la classe Etat (package state) nous permet de fabriquer pour chaque code tuile du fichier *« **map1.txt** »* le terrain correspondant, de créer un pointeur unique vers cet objet et de l'ajouter à la grille (tableau à deux dimensions contenant des pointeurs de Terrain).

Les personnages eux ne sont pas créés à partir d'un fichier, pour l'instant ils sont créés par la méthode initPersonnages de la classe Etat.

3.2 Conception logicielle

Le diagramme des classes pour le rendu général est présenté en Figure 6.

Classe Surface : cette classe possède deux attributs : une texture et un tableau de Vertex (quads) contenant la position des éléments et leurs coordonnées dans la texture. Elle possède les méthodes loadGrille et loadPersonnage lui permettant d'initialiser ses attributs à partir d'un tableau de Terrain ou d'une liste de Personnage et d'un fichier « *grille_tileset.png* » ou « *personnage_tileset.png* ». La méthode Draw a pour but de dessiner une texture pour ensuite permettre son affichage dans une fenêtre.

Classe TileSet : Cette classe possède plusieurs attributs : un id de type TileSetID, des entiers cellWidth et cellHeight (qui représentent respectivement la largeur et la longueur en pixel d'une tuile) et une chaîne de caractères imageFile (chemin vers un « *fichier.png* »). L'ID peut prendre différentes valeurs :

- GRILLESET
- PERSONNAGETILESET
- INFOSTILESET

Suivant l'ID passé en argument du constructeur de TileSet, les attributs cellWidth, cellHeight et imageFile sont initialisés différemment. Les méthodes getCellWidth, getCellHeight et getImageFile permettent de récupérer ces attributs.

Classe StateLayer : Cette classe possède un attribut etatLayer qui est une référence à une instance d'Etat. Elle possède également un tableau de pointeurs de TileSet et un autre de pointeurs de Surface. Le but de cette classe est de créer deux surfaces grâce à la méthode initSurface et d'initialiser leur texture. Cette classe est un observateur, elle implémente l'interface IObserver pour être avertie des changements d'état. Elle réagit ensuite en actualisant les textures.

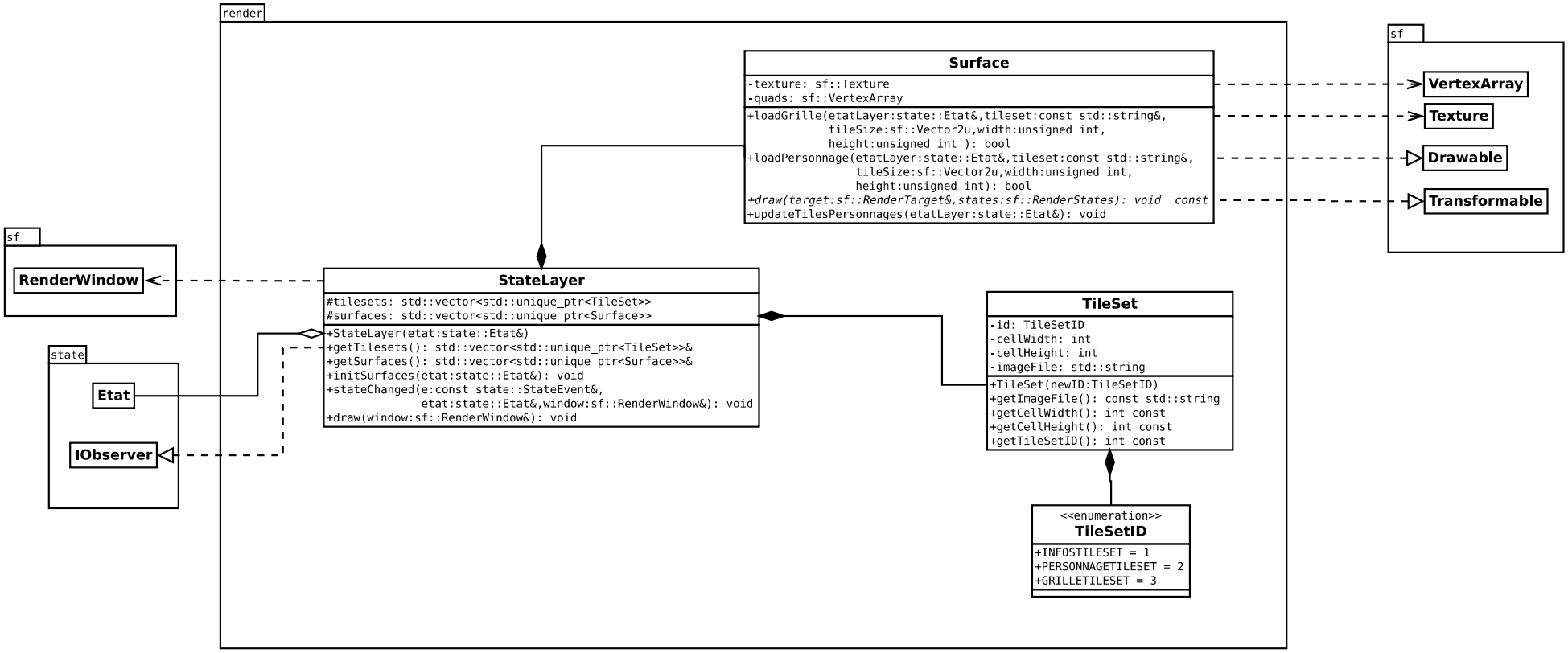

Figure 6 – Diagramme des classes de rendu

4 Règles de changement d’états et moteur de jeu

4.1 Règles de changement d’états

En début de tour, tous les personnages d’un joueur possèdent le statut DISPONIBLE. Ils peuvent donc tous être SÉLECTIONNÉS. Lorsqu’un personnage est SÉLECTIONNÉ par un joueur il peut effectuer selon les cas au moins une des 3 actions listées ci-dessous :

- Se déplacer
- Attaquer
- Terminer ses actions (son tour)

Un personnage ne peut se déplacer que d’une case par une case (sur des terrains praticables et inoccupés) . Il possède des points de mouvements qui correspondent au nombre de déplacement maximum qu’il lui est possible de faire. Lorsque tous ses points de mouvement ont été utilisés, il lui est impossible de continuer son déplacement.

Un personnage ne peut attaquer un autre personnage que lorsque celui ci se trouve dans son champ d’attaque et appartient au camp adverse (les attaques alliées ne sont pas autorisées).

Lorsqu’un personnage termine ses actions, son statut devient ATTENTE. Cela signifie qu’il ne pourra plus être SÉLECTIONNÉ (et ne pourra donc plus effectuer d’actions) avant le prochain tour du joueur.

L’action « Terminer son tour d’action » doit obligatoirement être effectuée par chaque personnage encore actif à chaque tour du joueur (même lors du dernier tour lorsque tous les adversaires ennemis sont morts, c’est ce qui déclenchera alors la victoire). Il existe 4 enchaînements d’actions possibles :

- Attaquer directement (lorsque cela est possible) ce qui termine automatiquement le tour d’un personnage
- Effectuer un ou plusieurs déplacements , attaquer puis terminer son tour automatiquement
- Effectuer un ou plusieurs déplacements puis terminer son tour manuellement
- Terminer son tour directement sans avoir effectuer aucune autre action

Le tour de jeu d’un joueur est terminé lorsque tous ses personnages sont en ATTENTE. C’est alors le tour du joueur adverse.

Lorsqu’un personnage est attaqué par un ennemi, il tente toujours une contre-attaque (à moins d’avoir été achevé durant la première attaque).

Si un personnage perd tous ses points de vie, son statut évolue et prend la valeur MORT.

Si tous les personnages d'un joueur meurent, la partie est terminée à la fin du tour adverse et le joueur adverse gagne.

Chaque action effectuée modifie l'état. Le choix du personnage sélectionné et des actions effectuées est provoqué par des commandes.

Pour l'instant les commandes sont écrites dans le fichier « ***main.cpp*** ».

4.2 Conception logicielle

Le diagramme des classes pour le moteur de jeu (« ***engine.dia*** ») est présenté en figure 7. Le moteur de jeu repose sur le Design Pattern Command.

Classe Moteur : C'est le cœur du moteur. Elle permet de stocker les commandes dans une std::map avec clef entière (avec « addCommande »). Ce mode de stockage permet d'introduire une notion de priorité : on traite les commandes dans l'ordre de leur clef (de la plus petite à la plus grande). Lorsque la méthode « update » est appelée, le moteur appelle la méthode « execute » de chaque commande puis supprime toutes les commandes une fois executées. La méthode « verificationDebutTour » change le joueur actif, réinitialise les points de mouvement de tous les personnages du joueur et procède à la récupération des PV des personnages se trouvant sur des cases MAISON ou FORTERESSE. La méthode « verificationFinTour » incrémente le nombre de tours si tous les personnages restants du joueur actif sont en ATTENTE et déclare la partie terminée si tous les personnages du joueur adverse sont MORTS.

Classes Commandes : Les classes Attaque, Deplacement et FinAction, héritant de la classe Commande, possèdent chacune une méthode « execute » qui fait effectuer à un personnage l'action correspondante.

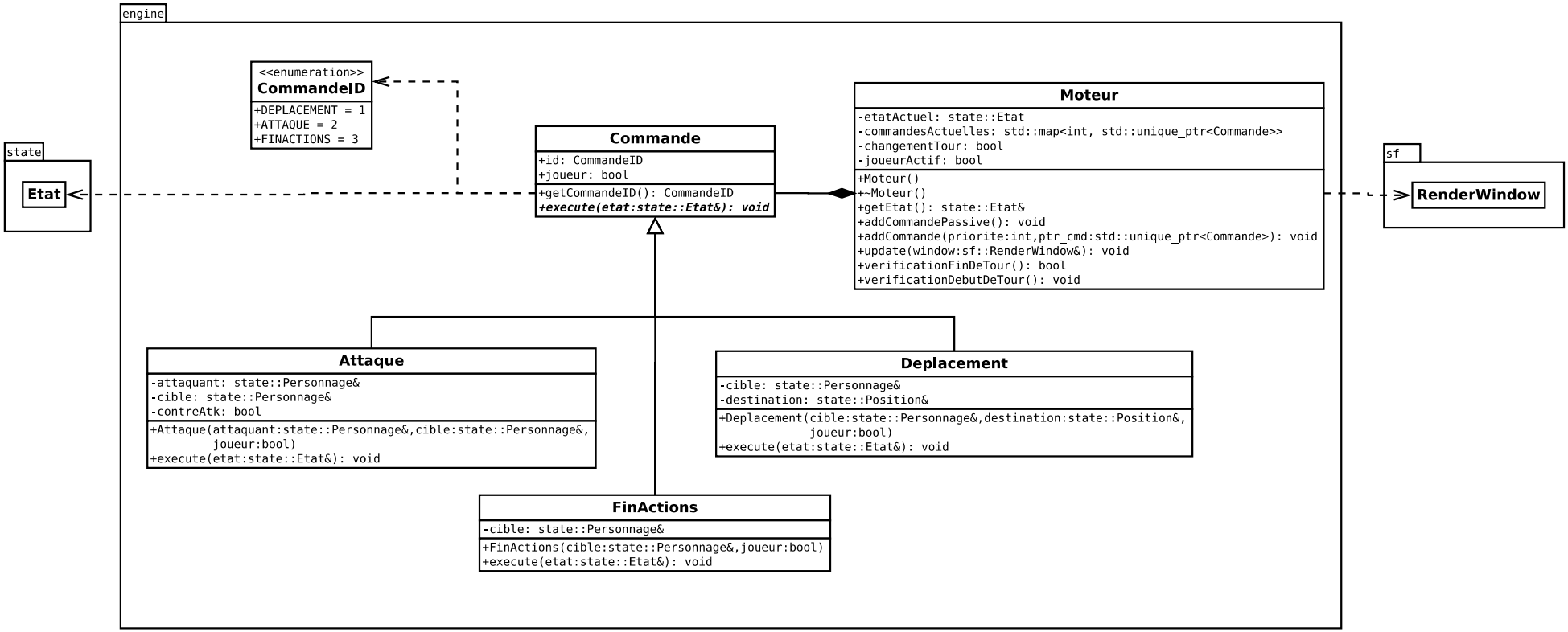

Figure 7 – Diagramme des classes de moteur