Projet Advance Wars

Moothery Franck
Boutaleb Yousr

Livrable 2.2

Tables des matières

1. 1 Présentation général

1.1 Archétype

L'objectif de ce projet est de réaliser le jeu Advance wars.

1.2 Règle du jeu

Le joueur gagne la partie s'il capture la base adverse. Le jeu se déroule en tour par tour. Il gagne par tour une somme d'argent définie qu'il doit investir pour acheter des unités de combat. La possibilité de capturer des bâtiments est possible pour améliorer son armée, ses gains ou son avantage du terrain. Il peut y avoir jusqu'à quatre personnes qui jouent simultanément dans une partie.

1.3 Ressources

L'affichage repose sur quatre textures

Figure 1: Texture soldat

Figure 2: Texture bâtiment

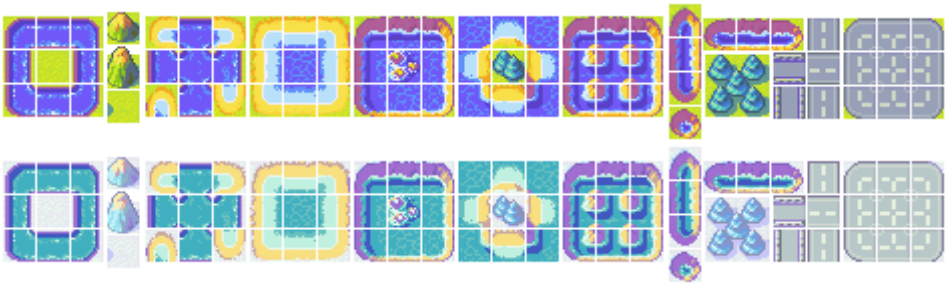


Figure 3: Texture carte

Figure 4: Texture icone

2. Description et conception des états

2.1 Description des états

Un état du jeu est formé par un ensemble d'éléments fixes (la carte) et un ensemble d'éléments mobiles (les soldats). Tous les éléments possèdent les propriétés suivantes :

— Coordonnées (x, y) dans la grille
— Identifiant de type d'élément : ce nombre indique la nature d'élément (ie classe)

2.1.1 Etat éléments fixes

La carte est formée par une grille d'éléments nommé « cases ». La taille de cette grille est fixée au démarrage de la partie. Les types de cases sont :

Cases « Eau ». Les cases « eau » sont des éléments infranchissables pour les éléments mobiles terrestres. Le choix de la texture est purement esthétique, et n'a pas d'influence sur l'évolution du jeu.

Cases « terrain ». Les cases « terrain » sont les éléments franchissables par tous les éléments mobiles. On considère les types de cases « terrain » suivants :

— Les espaces « herbe », qui contiennent de l'herbe
— Les espaces « route », qui contiennent une route
— Les espaces « sable », qui contiennent du sable

Cases « Rocher ». Les cases « rocher » sont les éléments franchissables par certains éléments mobiles.

Cases « Bâtiment ». Les cases « bâtiment » sont les éléments franchissables par tous éléments mobiles. Elle permet aussi d'utiliser certaine fonctionnalité comme modifier des éléments du jeu (caserne, QG, bâtiment). On considère les types de cases « bâtiment » suivants :

— Les espaces « bâtiment »
— Les espaces « caserne », qui contiennent une caserne qui permet de créer des unités
— Les espaces « QG », qui contient un QG. C'est le bâtiment à capturer pour gagner la partie.

Suivant l'élément fixe nous avons un niveau de défense qui va s'appliquer à l'unité.

2.1.2 Etat éléments mobiles

Les éléments mobiles possèdent un mouvement, des points de vie, sa défense, sa puissance et une position. Un élément mobile doit être forcément sur une case. Il ne peut pas être entre deux cases. Ces éléments sont dirigés par le joueur, qui commande la propriété de direction.

Elément mobile « infanterie ». On utilise une propriété que l'on nommera « statut », et qui peut prendre les valeurs suivantes :

— Statut « mouvement en attente » : cas où l'infanterie est en attente d'action
— Statut « mouvement fini » : cas où l'infanterie à finit de faire son mouvement
— Statut « capture » : cas où l'infanterie capture un bâtiment

Elément mobile « tank ». Il possède des plus de points de vie que l'infanterie, il peut se déplacer sur un plus grand nombre de case et peut attaquer de loin avec une portée maximale précise. « Statut » peut prendre les valeurs suivantes :

— Statut « mouvement en attente » : cas où l'infanterie est en attente d'action
— Statut « mouvement fini » : cas où l'infanterie à finit de faire son mouvement

Elément mobile « Hélicoptère ». Il est semblable au tank sauf qu'il peut se déplacer au-dessus de tous les éléments fixes. « Statut » peut prendre les valeurs suivantes :

— Statut « mouvement en attente » : cas où l'infanterie est en attente d'action
— Statut « mouvement fini » : cas où l'infanterie à finit de faire son mouvement

2.1.3 Etat général

A l'ensemble des éléments statiques et mobiles, nous rajoutons la propriété de tour pour chaque joueur. Cela va permettre de donner la main au joueur qui devrait jouer.

Le diagramme des classes pour les états est présenté en page 7, dont nous pouvons mettre en évidence les groupes de classes suivants :

Classes Élément. Toute la hiérarchie des classes filles d'Elément permettent de représenter les différentes catégories et types d'élément :

— Eléments mobiles
— Eléments fixes

Conteneurs d'élément. Viennent ensuite les classes State () et ElementTab et ElementChar qui permettent de contenir les éléments instanciés. ElementTab est un tableau en deux dimension qui contiendrait les éléments statisques, et ElementChars contiendrait les éléments mobiles (ils contiennent une liste de pointeur qui va pointer sur chaque objet instancier). Enfin, la classe State est le conteneur principal, à partir duquel on peut accéder à toutes les données de l'état (on n'a pas encore défini toutes ces méthodes).

Méthode du Conteneurs ElementTab :

— Deux constructeurs pour créer différente taille de liste
— SetElement : ajoute un élément en bout de liste
— chgList : ajoute un élément à l'endroit défini si la case est NULL
— chgList2 : interverti deux éléments dans la liste
— createElementcsv : crée une liste d'unique ptr<Element> à partir d'un fichier csv. Le fichier csv est traduit en une liste qui à partir de la valeur des éléments de la liste instancie un Element quelconque (CHAMPDEBATAILLE, HERBE, EAU, …)
— ElementTocarte : permet de créer une liste d'entier à partir de la liste d'unique ptr<Element>

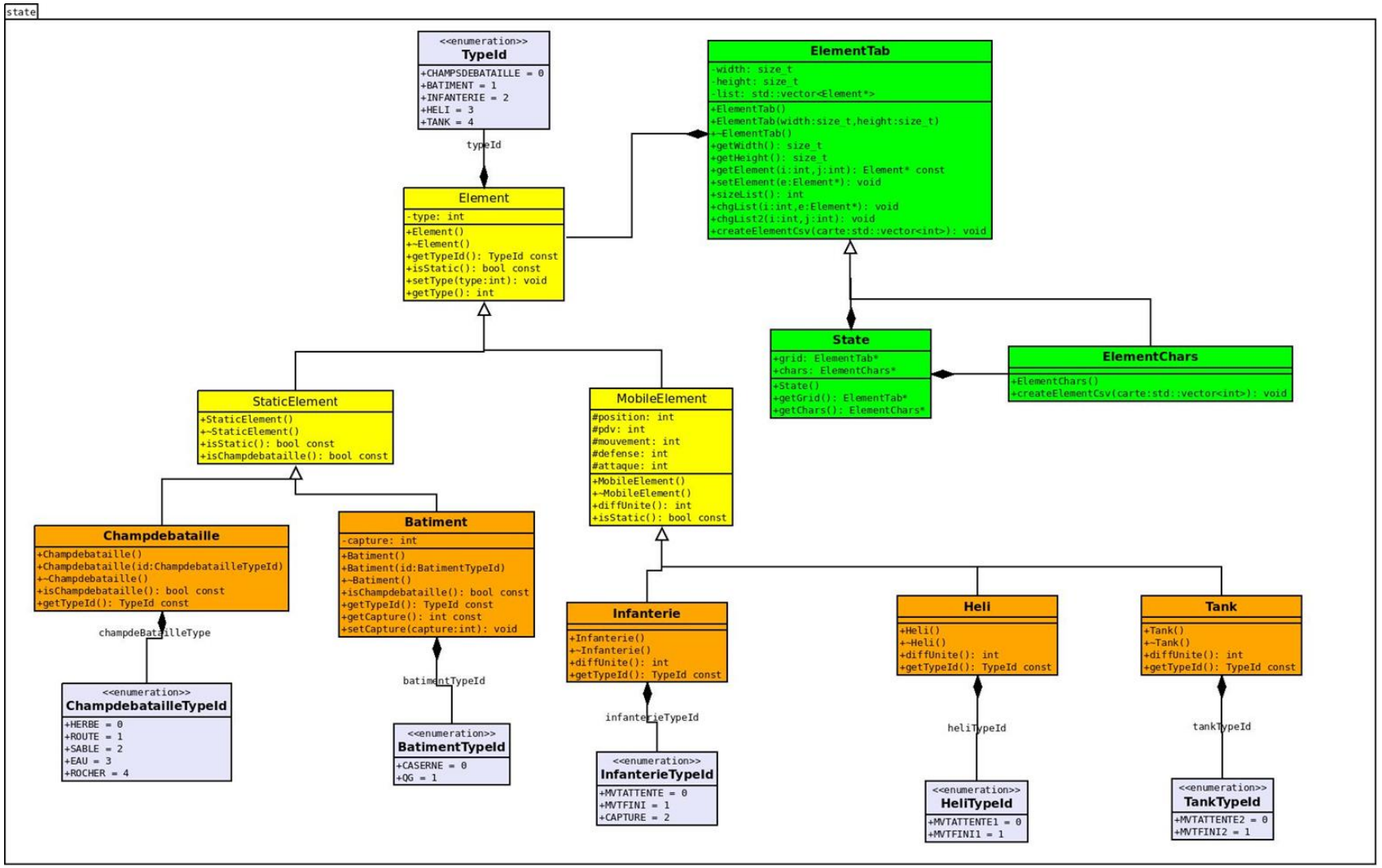


Figure 5 : Diagramme des classes d'état

3. Rendu : Stratégie et Conception

3.1 Stratégie de rendu d'un état

Pour le rendu d'un état, Une stratégie de bas niveau a été réalisé. Le rendu d'un état est la superposition de deux plans. Un premier plan est utilisé pour afficher les éléments du décor et un deuxième plan est utilisé pour afficher les éléments mobiles. L'affichage du premier plan est réalisé par le chargement d'un fichier « .csv » qui contient des chiffres entre 0 et 4. Pour l'affichage du deuxième plan, la liste d'Element qui est rempli avec des éléments ou des éléments null est utilisé pour créer une liste de chiffres entre 0 et 2. Les chiffres serviront à différencier tous les types d'Element mobile. De la même manière que le premier plan le deuxième plan est géré par Tilemap. Une classe Tile est utilisé pour ranger toutes les informations des images. Cette classe n'est d'ailleurs pas encore utilisée pour des problèmes d'implémentations.

3.2 Conception logiciel

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

Méthode de Tilemap :
— lirefichiercsv : Cette fonction renvoie une carte (liste) généré à partir d 'un fichier csv
— load : cette fonction charge la texture puis redimensionne le tableau de vertex pour qu'il puisse contenir tout le niveau, ensuite elle remplit le tableau de vertex avec un quad par tuile, et enfin définit les coins et les coordonnées de texture.

Méthode de GridTileSet/CharsTileSet :
— GridTileSet : remplit la liste des informations nécessaires
— getTile : permet de renvoyer la tuile correpondante à l'élement envoyé.
— getImageFile : renvoi le chemin d'accès à l'image

Methode de Layer :

— initmap : initialise Tilemap pour afficher une carte d'élément statique
— initRandMap : initialise Tilemap pour afficher une carte d'éléments statiques aléatoirement
— displayChars : initialise Tilemap pour afficher des éléments mobiles

3.3 Description des fonctions tests

Après l'implémentation de chaque méthode, on utilise des fonctions test qui vérifie le bon fonctionnement de nos méthodes, on retrouve :

Fonction touteslesfonctions:
— teststate : Elle permet de faire le test du livrable 1.finale.
— testrender : Elle permet de faire le test du livrable 2.1
— testengine : Elle permet de faire le test du livrable 2.2
— testenginerender : Elle permet de faire le test du livrable 2.2 avec un affichage

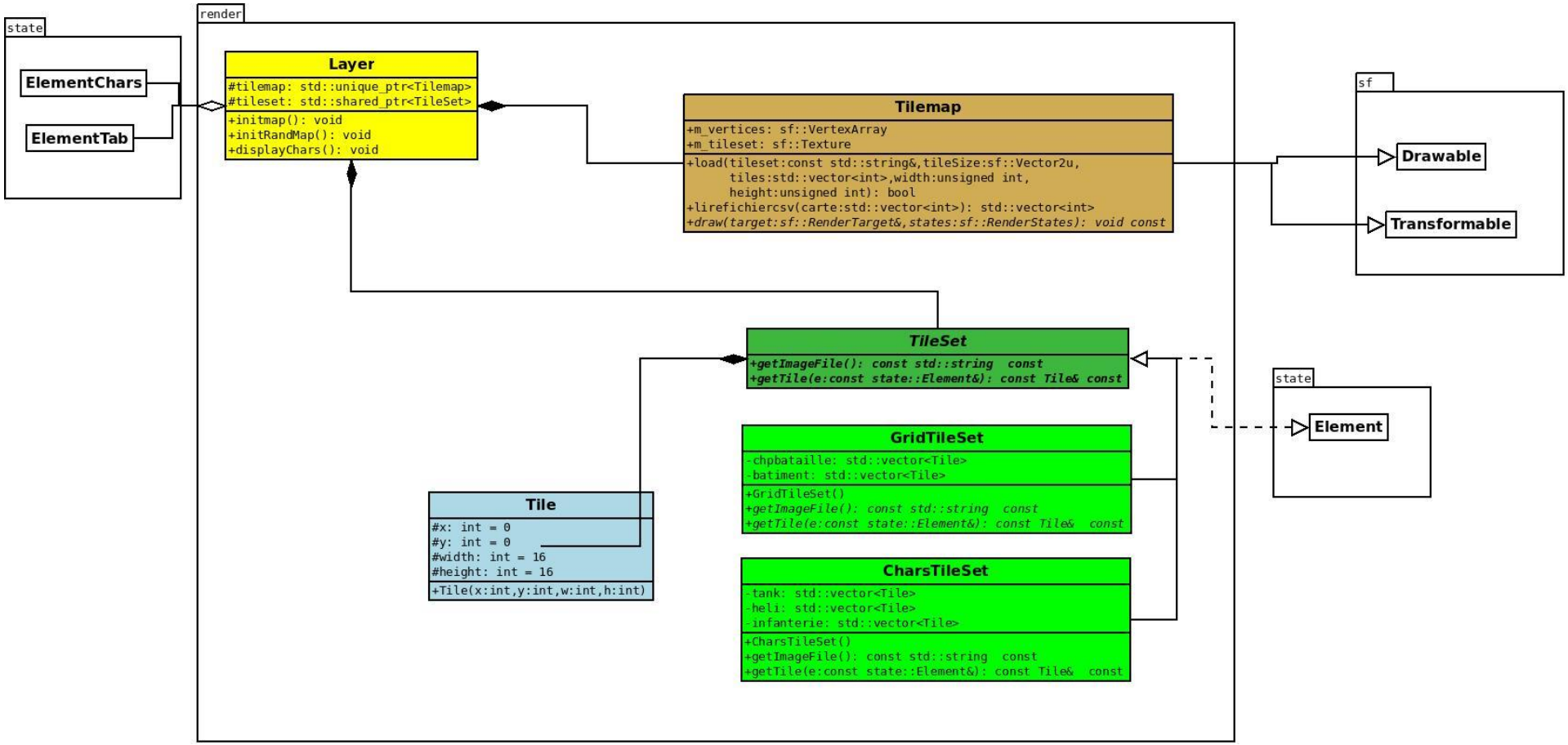


Figure 6 : Diagramme du render

4. Règles de changement d'états et moteur de jeu

4.1 Changements

Les changements extérieurs sont provoqués par des commandes extérieurs :

— Commandes principales : « Charger un niveau » : On fabrique un état initial à partir d'un fichier
— Commandes bouger un personnage : On déplace le personnage désiré sur une case désirée s'il n'y a pas un autre personnage dessus ou s'il peut se déplacer sur ce type de terrain
— Commandes attaquer un personnage : On attaque le personnage désiré avec un autre personnage qui appartient à un autre joueur. Si un personnage à ses points de vie réduit à 0, il se voit détruit
— Commandes capture un bâtiment : On capture un bâtiment avec une infanterie qui est placée sur celui-ci
— Commandes créer un personnage : On créer un personnage avec un bâtiment et celui-ci apparait sur le bâtiment

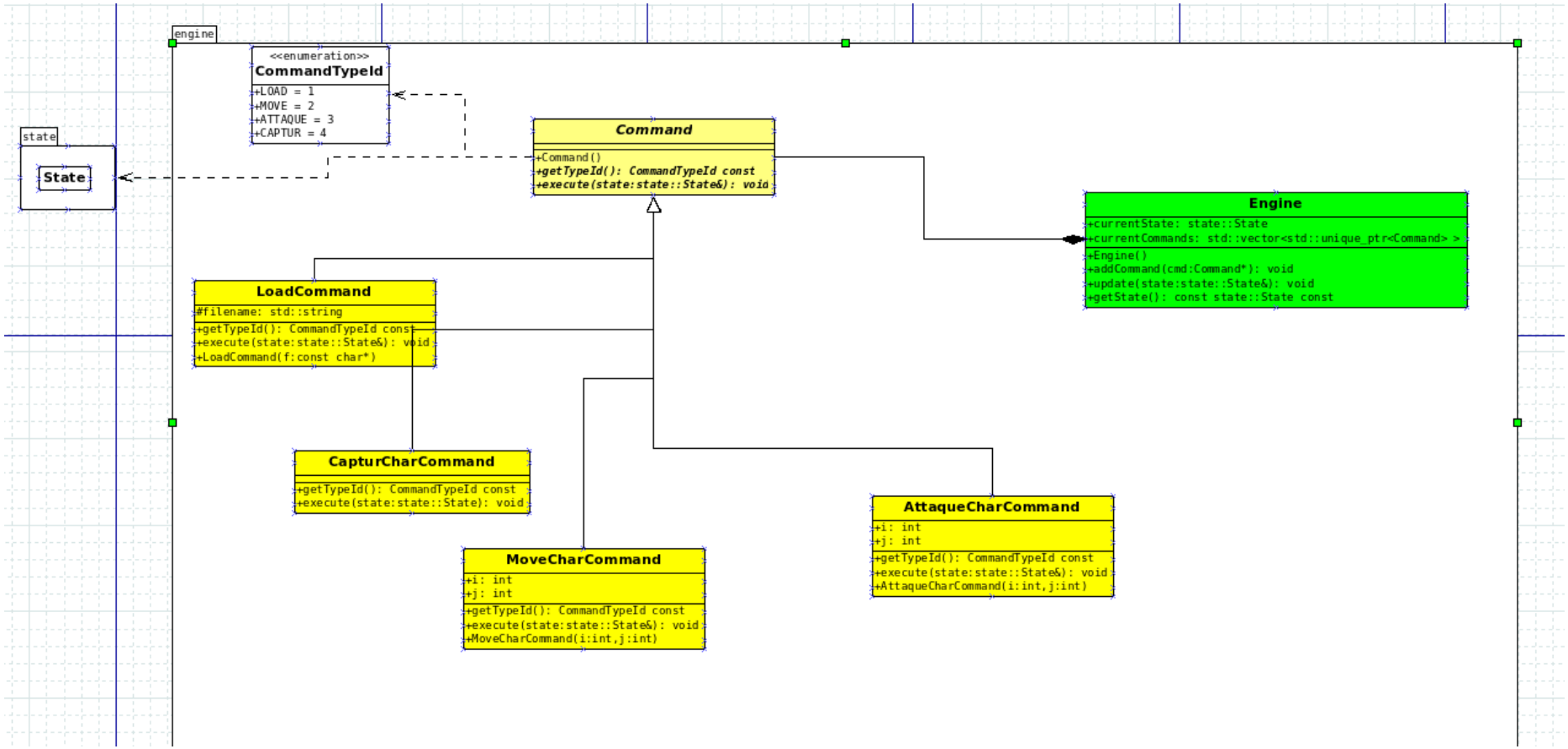
5. Conception logiciel

On utilise le patron de conception Command pour appliquer des commandes à notre état.

Classes Command :

— MoveCharCommand : Cette fonction permet de déplacer une unité
— LoadCommand : Cette fonction permet de créer nos éléments statiques
— AttaqueCharCommand : Cette fonction permet à une d'en attaquer une autre
— createCharCommand : Cette fonction permet à un bâtiment de créer une unité

Classes Élément. Engine nous permet de stocker les commandes à exécuté puis à les utiliser avec la méthode update.


engine
<<enumeration>>
CommandTypeId
+LOAD = 1
+MOVE = 2
+ATTAQUE = 3
+CAPTUR = 4
state
State
Command
+Command()
+getTypeId(): CommandTypeId const
+execute(state:state::State&): void
Engine
+currentState: state::State
+currentCommands: std::vector<std::unique_ptr<Command> >
+Engine()
+addCommand(cmd:Command*): void
+update(state:state::State&): void
+getState(): const state::State const
LoadCommand
#filename: std::string
+getTypeId(): CommandTypeId const
+execute(state:state::State&): void
+LoadCommand(f:const char*)
CapturCharCommand
+getTypeId(): CommandTypeId const
+execute(state:state::State): void
MoveCharCommand
+i: int
+j: int
+getTypeId(): CommandTypeId const
+execute(state:state::State&): void
+MoveCharCommand(i:int,j:int)
AttaqueCharCommand
+i: int
+j: int
+getTypeId(): CommandTypeId const
+execute(state:state::State&): void
+AttaqueCharCommand(i:int,j:int)