FAUJEAU François
BALACHANDRAN Chirojean
3A IS

PROJET CIVILIZATION

Rapport du projet logiciel transversal

Année scolaire : 2018-2019

Table des matières

1 Présentation générale

1.1 Archétype

Le but de notre projet logiciel transversal est de créer une version très simplifiée du jeu civilization. Ce jeu est un jeu de stratégie tour par tour où l'objectif d'un joueur est de développer un empire en choisissant une civilisation au choix parmi plusieurs.

FIGURE 1 – Image d'un gameplay de civilization 3

1.2 Règles du jeu

Le but pour chaque joueur est de développer un empire avec la civilisation romaine. Pour cela, le joueur doit améliorer et gérer ses ressources (mines d'or, terres agricoles) ainsi que ses bâtiments (caserne et palais).
Nous avons choisi que chaque joueur pourrait seulement utiliser la civilisation romaine. Chaque empire permet de créer différents bâtiments :

1. Le palais : bâtiment principal de l'empire. Son niveau correspond au niveau de l'empire. Plus sont niveau est élevé et plus il est possible d'améliorer les autres bâtiments et unités.
2. La caserne : permet de former des soldats. Quand le niveau de la caserne augmente, le niveau des troupes formées et le nombre des troupes qui peuvent être formées augmentent. Il y aura quatre types d'unités de combat : un cavalier, un épéiste, un archer et une catapulte. A la création d'une unité, le joueur choisira si elle est créée pour défendre l'empire ou si elle peut se déplacer sur la carte de jeu.
3. Ressources : le joueur possèdera trois types de ressources à savoir les mines d'or, la réserve de bois et les terres agricoles. Quand le niveau d'une ressource

augmente, la capacitée de stockage de cette ressource et la quantité produite sont augmentées.

Pour améliorer les bâtiments, il faudra que le joueur dépense une certaine quantité de bois et d'or et pour la formation de soldats une certaine quantité de récoltes agricoles et d'or.
Pour développer son empire plus rapidement il est possible d'en combatre un autre (détenu soit par l'IA, soit par un joueur) ou de s'en défendre. En effet, en cas de victoire l'empire victorieux reçoit une partie des ressources de l'empire adverse et un bonus de victoire. L'empire qui a perdu recupère, lui, un faible bonus de défaite.
Le système de combat est le suivant. Chacun leur tour, les joueurs pourront décider de :

— se déplacer pour se rapprocher du combat.
— choisir qu'un soldat en attaque un autre s'il est dans sa zone d'action.

Un combat se termine lorsque tous les soldats d'un joueur sont éliminés.
Dans notre jeu, un tour correspond à une action : une amélioration de bâtiment, la formation d'une unité, son déplacement.

1.3 Ressources

Voici les différentes ressources (images, textures, sprites) utilisées pour le développement du jeu. Pour créer les cartes de jeu nous les génèrerons de manière aléatoire en mode isométrique. Ces cartes sont des cartes de jeu 25 x 25.

FIGURE 2 – Bâtiments de la civilisation romaine en fonction des niveaux

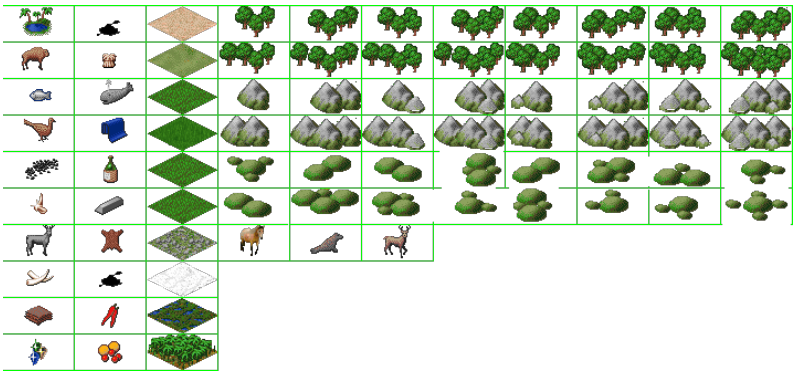

FIGURE 3 – textures pour la création de la map

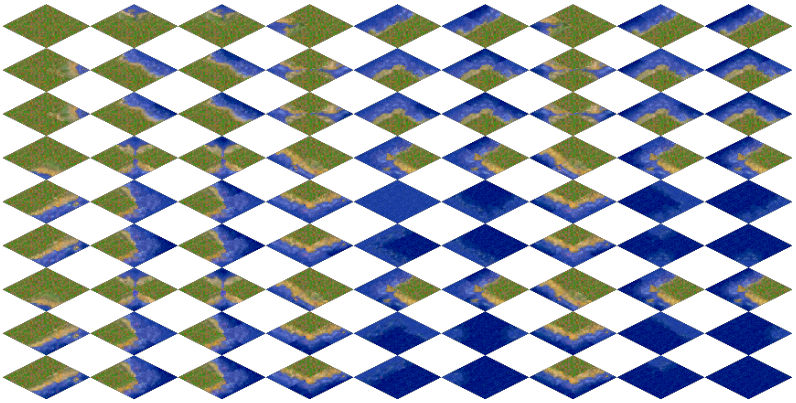

FIGURE 4 – textures pour la mer sur la map

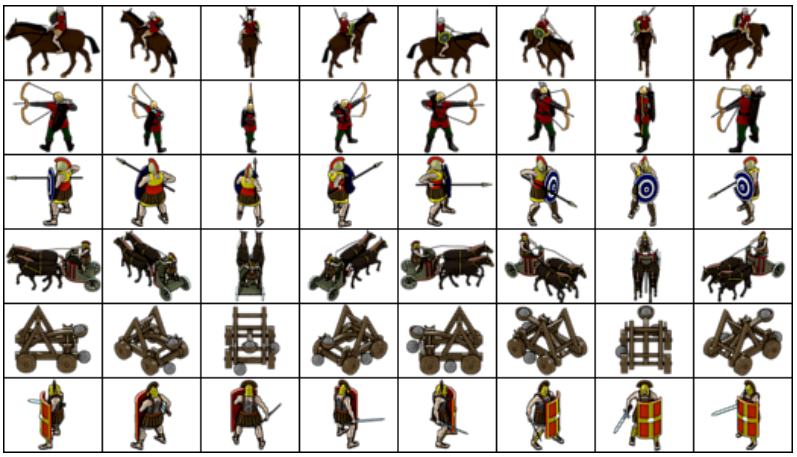

FIGURE 5 – images des unités de combats

2 Description et conception des états

2.1 Description des états

Un état de jeu est constitué d'une carte sur lequel se situent différents empires avec leurs différents bâtiments (éléments fixes), les éléments de décor (éléments fixes) et des unités de combats (éléments mobiles).

2.1.1 Description de la carte

La carte est représentée avec une grille composée de cases contenant des éléments sur chacunes d'elles avec une certaine position. Elle est composée d'empires, qui sont eux-mêmes constitués de bâtiments, et d'éléments comme les bâtiments, les unités de combats et les éléments de décor. Certains éléments sont franchissables par les unités et d'autres non. La taille de la carte est fixée.

2.1.2 Description des états éléments fixes

Les éléments fixes sont les décors et les bâtiments et les empires qui sont abstraits. Tous les éléments fixes possèdent une position (x,y) entre 0 et 24.

Décors

Les décors sont de différents types : eau, herbe, dunes, montagnes, arbre, animaux... Ils ont chaucun un identifiant. Les décors possèdent un attribut passable qui permet de savoir si l'élément est franchissable ou non par les unités de soldats.

Empire

Un empire a plusieurs attributs :
— un identifiant pour le différencier des autres empires (chiffre)
— un nom
— un niveau compris entre 1 et 4
— des points de vie qui sont égaux à ceux du palais
— des ressources (or, bois, nourriture) pour se développer et former des troupes
— une liste de position permettant d'indiquer où se positionne l'empire sur la carte

Enfin, il est composée d'une caserne, d'un bâtiment ressource et enfin d'un palais.

Bâtiments

Les bâtiments ont tous ces attributs :
— un identifiant pour le différencier des autres (chiffre)
— un niveau qui va de 1 à 4
— un coût en bois et en or pour les construire
— un identifiant pour leur texture graphique

Il y a trois types de bâtiments : la caserne, les ressources et le palais.

La caserne a une capacité maximale de formation de troupes et possède comme attribut le nombre de troupes formées. Elle permet de former les différentes troupes.

Les ressources ont un niveau de production identique pour l'or, le bois ainsi que la nourriture. Ils vont permettre d'augmenter la richesse d'un joueur à chaque tour.

Le palais quant à lui possède des points de vie. C'est lui qui sera attaqué par un joueur et sa destruction mènera à la fin d'une partie.

Une carte de jeux typique sera composée de 25 x 25 cases disposées de manière isométrique. Ces cartes seront générées de manière aléatiores.

2.1.3 Description des états éléments mobiles

Tous les éléments mobiles possèdent une position (x,y). Ils possèdent tous :

— un identifiant pour les différencier
— un niveau de vie
— un niveau de dommages en attaque
— une portée pour attaquer nombre de déplacements de cases limité
— un coût en or et en nourriture pour la former
— un identifiant de texture graphique.

Les unités de soldats dans le jeu sont les unités de combats : décurion, cavalier, catapulte et archer.

2.2 Conception des états

Dans cette partie nous allons décrire le diagramme UML d'état du jeu. (Voir figure 6)

Class "Element" : Cette classe est la classe mère de tous nos éléments, mobiles ou statiques. Nous avons choisi pour cette classe qu'elle contienne une position, la position de notre élément sur la map, et une méthode isPassable(), qui nous permet de savoir si l'élément peut être traversé par un autre.

Class "Units" : Cette classe est la classe mère de toutes nos unités mobiles sur la carte. Elle hérite d'"Element". Elle contient tous les paramètres nécessaires à la création d'une unité (vie, attaque, niveau, ...).

Class "Arrow" : Cette classe est la classe fille de Units. Elle hérite d'"Element". Elle permet de construire des unitées de type archer avec leurs attributs initialisés en fontion de leur niveau. Ce sera la façon la plus utilisée de créer un archer. Il existe de même des classes pour Decurion, Cavalier et Catapult.

Class "Buildings" : Cette classe est la classe mère de toutes nos unités statiques sur la carte. Elle hérite d'"Element". Elle contient tous les paramètres nécessaires à la création d'un bâtiment (niveau, coût, ...).

Class "Barrack" : Une caserne est un bâtiment et hérite donc de "Buildings", elle ajoute en plus quelques fonctionnalitées telles que la création d'unités. De même pour les bâtiments de type ressource et palais.

Concernant les coûts, nous avons créé deux classes qui permettent de créer le coût des unités et des bâtiments, et de pouvoir les modifier.

Nous avons aussi choisi d'attribuer à chaque élément un id qui permet de savoir instantanément lequel il est.

Class "Decor" : Cette classe hérite d'élément, elle permet de créer les différents décors qui seront présents sur la carte de jeu.

Class "Empire" : Dans notre jeu nous avons besoin à tout instant d'avoir une classe qui stocke les éléments importants d'un empire. Par exemple un empire à un nom, une vie, de l'or, du bois, de la nourriture, ... Ainsi, cette classe va agir comme un conteneur de tous les élements clés de notre empire.

Class "Map" : cette classe contient les éléments sur la carte de jeu. Elle permet d'ajouter des éléments, de les récupérer ou d'en supprimer.

Class "Observable" : cette classe permet de réagir au des actions gérées par le moteur de jeu. Par exemple, lorsqu'un joueur souhaite améliorer un bâtiment ou créer une unité sur la carte de jeu, il faut que les cartes contenues dans notre classe Map soient modifées en conséquence.
Elle contient une méthode appelée "notifyObserveur" qui va notifier ses observeurs. Cet observable est composé de deux observeurs : UnitsObserver et BuildingsObserver.

Class "UnitsObserver" : Cette classe est un Observer sur les Units. Elle va permettre de réaliser tous les changements nécessaires sur le state des units demandés par le moteur.

Class "BuildingsObserver" : Cette classe est un Observer sur les Buildings. Elle va permettre de réaliser tous les changements nécessaires sur le state des buildings demandés par le moteur.

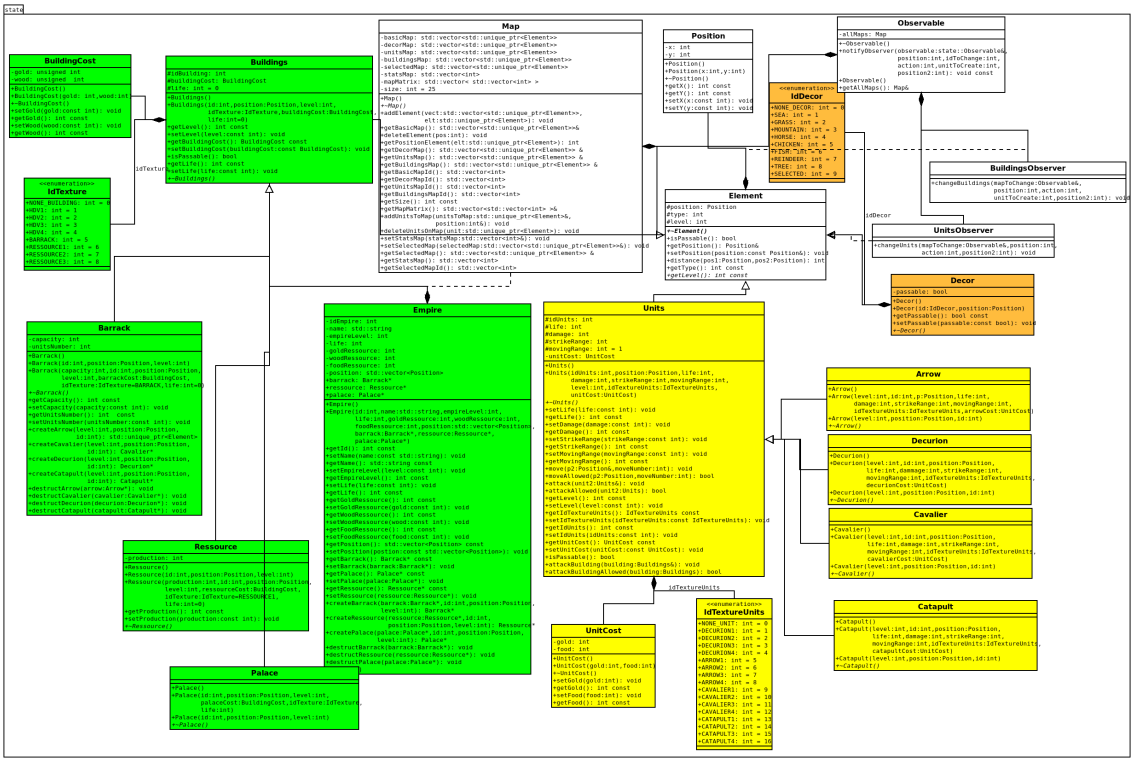

FIGURE 6 – Diagramme de classes d'état

,

3 Rendu : Stratégie et conception des états

3.1 Stratégie de rendu d'un état

Pour le rendu des états, nous avons décidé de faire une carte en 2D avec des tuiles et en isométrie. Cette carte sera à chaque fois générée aléatoirement et donc à chaque lancement du programme elle sera différente. Nous utilisons la librairie SFML. Nous avons trois tilesets qui vont nous servir à créer les textures de la carte :

— un pour les textures du terrain (eau, herbe, animaux ...)
— un pour les bâtiments (caserne, ressource, palais)
— un pour les unités (archer, cavalier, decurion, catapulte)

Sur le côté gauche, un petit menu permet de donner le niveau, la vie et les dommages s'il y en a des éléments.

Nous aurons une carte divisée en quatre plans :

— un pour l'herbe et l'eau qui sont nos éléments visuels de base
— un pour les éléments de décor comme les montagnes, les arbres ou encore les animaux
— un pour les bâtiments
— un pour les unités mobiles

Les trois premiers plans seront initialisés au départ du jeu grâce aux textures et à des tableaux constitués des numéros de texture d'éléments, selon une position, générés aléatoirement, selon certaines règles de création de map :

— Toute la carte est préalablement constitué d'herbe seulement
— Ensuite, trois zones d'eau sont tirées au sort aléatoirement
— Puis, trois zones d'arbres et de montagnes sont tirées aléatoirement et quelques animaux sont placés
— Enfin, trois positions pour les empires sont tirées au sort et les trois bâtiments sont placés côte à côte

Tous les plans seront ensuite superposés pour former la carte. Les troix premiers plans seront fixes et seul le plan des unités changera. Ce dernier sera en effet modifié à chaque tour lorsque l'utilisateur fera déplacer ses unités mobiles. La mise à jour de l'affichage et du changement d'états se fera à une fréquence de quelques hertz.

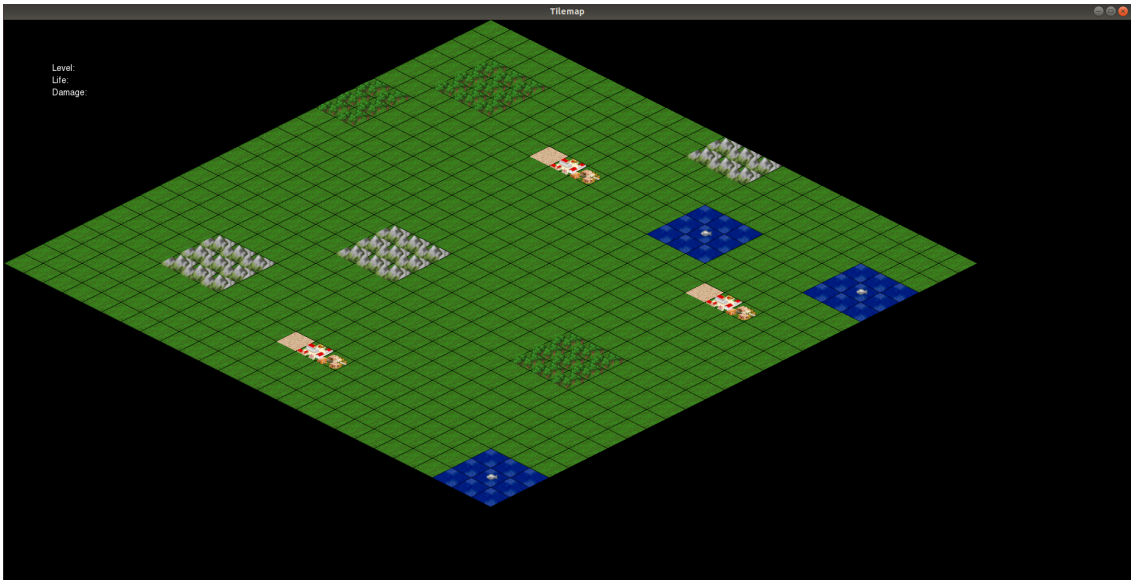

FIGURE 7 – Exemple d'un état généré aléatoirement

3.2 Conception de rendu d'un état

Dans cette partie nous allons décrire le diagramme UML de rendu d'un état du jeu.

Class "Tiles" : C'est un élément de base des tilesets. Ils ont une longueur et une largeur comme attributs ainsi qu'une position dans le tileset.

Class "TileSet" : Cette classe est la classe mère de tous les tilesets (tableaux de tuiles associés aux éléments). Elle est composée de tuiles. Nous avons choisi pour cette classe qu'elle contienne une méthode pour connaître la largeur d'une tuile et une autre qui permet de renvoyer la hauteur d'une tuile.

Class "BuildingTileSet" : Elle hérite de "TileSet". Elle contient trois tableaux de tuiles comme attributs (un pour les casernes, un pour les palais et un pour les ressources) permettant de donner la texture associée à chaque élément. Elle possède une méthode permettant de renvoyer le chemin du fichier png contenant le tileset et une autre pour obtenir la tuile d'un bâtiment donné.

Class "DecorTileSet" : Elle hérite de "TileSet". Elle contient un tableau de tuiles pour les décors comme attributs permettant de leur donner la texture associée. Elle possède une méthode permettant de renvoyer le chemin du fichier png contenant le tileset et une autre pour obtenir la tuile d'un décor donné.

Class "UnitsTileSet" : Elle hérite de "TileSet". Elle contient quatre tableaux de tuiles comme attributs (un pour les archers, un pour les cavaliers, un pour les décurions et un pour les catapultes) permettant de donner la texture associée à chaque unité. Elle possède une méthode permettant de renvoyer le chemin du fichier png contenant le tileset et une autre pour obtenir la tuile d'une unité donnée.

Class "MapCreator" : Cette classe permet de créer une carte avec des textures d'un tileset. Elle constitue les plans d'une carte. Elle contient des textures et un tableau en attributs. Elle a des méthodes pour charger un fichier de texture, d'initialiser lee tableau, pour positionner un sprite sur une carte et lui donner sa texture et enfin d'afficher cette carte.

Class "MenuLayer" : Cette classe permet le menu permettant d'indiquer le niveau, la vie et les dommages.

Class "Layer" : Cette classe permet de créer des plans de carte. Elle est composée de MapCreator. Elle a comme attributs les tableaux de textures, selon leurs position, des éléments de base (herbe, eau), des décors et des bâtiments. Elle fait le lien entre la map du jeu contenue dans le diagramme d'état et le rendu.

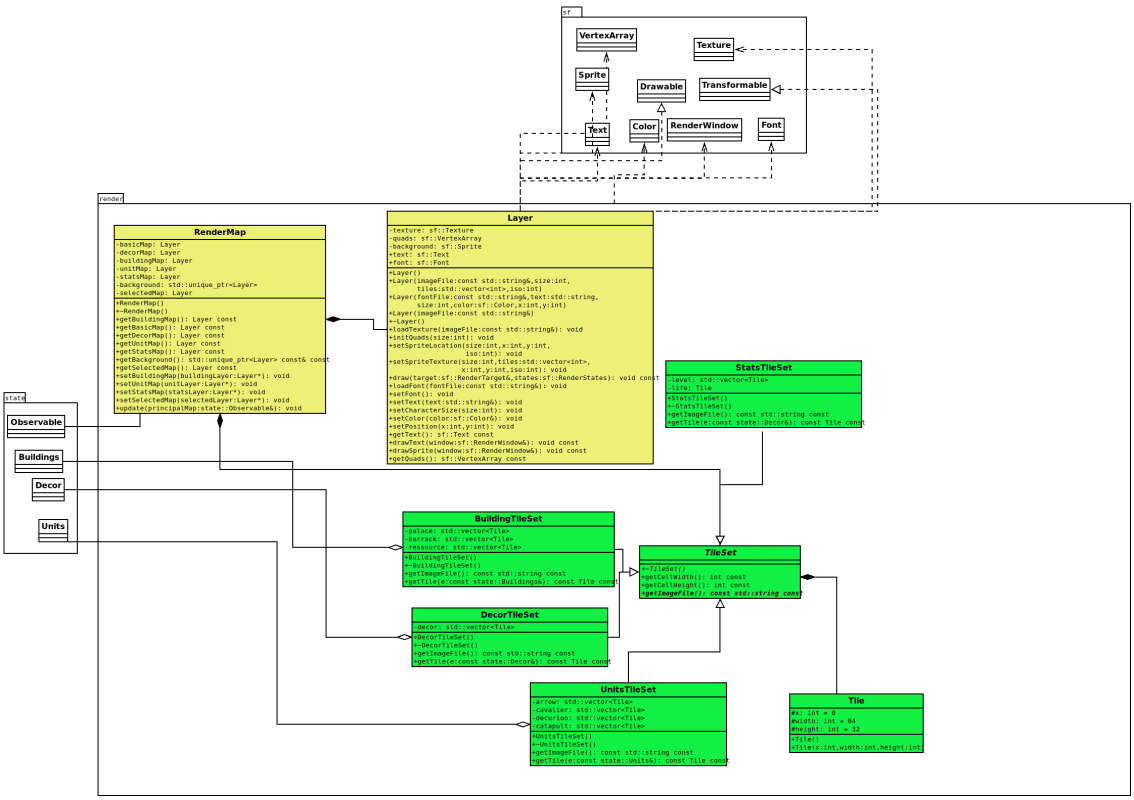

FIGURE 8 – Diagramme de classes du rendu

4 Règles de changements d'états et moteur de jeu

4.1 Horloge globale

Tous les changements dans le jeu suivent une horloge globale pour le passage d'un état à un autre.
Ces changements sont réalisés en fonction du temps de passage d'un état à un autre. On attend la fin du changement d'état pour en réaliser un nouveau.

4.2 Changements extérieurs

Les changements extérieurs sont générés par des cliques du joueur. Les commandes sont les suivantes :

- Cliquer sur n'importe quelle texture et consulter les actions qui lui sont associées ainsi que ses attributs (niveau de vie avec les coeurs, niveau, capacité ...).
- Augmenter le niveau des bâtiments.
- Créer des unités en séléctionnant leurs positions initiales avec un clique.
- Déplacer des unités en choisissant les cases parmi celles possibles.
- Faire attaquer des unités entre elles.
- Attaquer un hôtel de ville.

4.3 Changements autonomes

Ces changements sont ceux qui sont gérés par le moteur sans être provoqués par des actions utilisateurs

- Augmentation des ressources à chaque tour.
- Disparition des unités qui n'ont plus de vie ainsi que de l'empire du palais n'ayant plus de vie.
- Règles de mouvement et d'attaque.
- Affichage des statistiques des bâtiments et des unités.

4.4 Conception logiciel

Le diagramme des classes du moteur est en figure 9 et diponible dans le dossier src du projet sous le nom de engine. Dans cette étape de conception logiciel nous avons utilisé un pattern command. A son exécution, le moteur appelle des commandes en fonction des actions utilisateur.

Classes de commandes : Elles héritent toutes de la classe Command. Les commandes suivantes sont disponibles :

- CaseIdentifier : Permet d'identifier la case sur laquelle on se trouve.
- Possibilities : Permet de déterminer quelles sont les actions possibles à partir de cette case.
- PrintStats : Affiche les stats (vie, niveau, bouttons) des joueurs et des bâtiments.
- LevelUp : Augmente le niveau d'un bâtiment.

— CreateUnit : Créer une unité à partir d'un premier clique sur une caserne. Le second clique permet de positionner l'unité crée sur la map. Des boutons seront disponibles pour choisir le type d'unités à créer.
— Move : Permet de déplacer une unité
— Attack : Permet d'attaquer une unité ou un hôtel de ville.
— Engine : implémente le moteur de jeu.

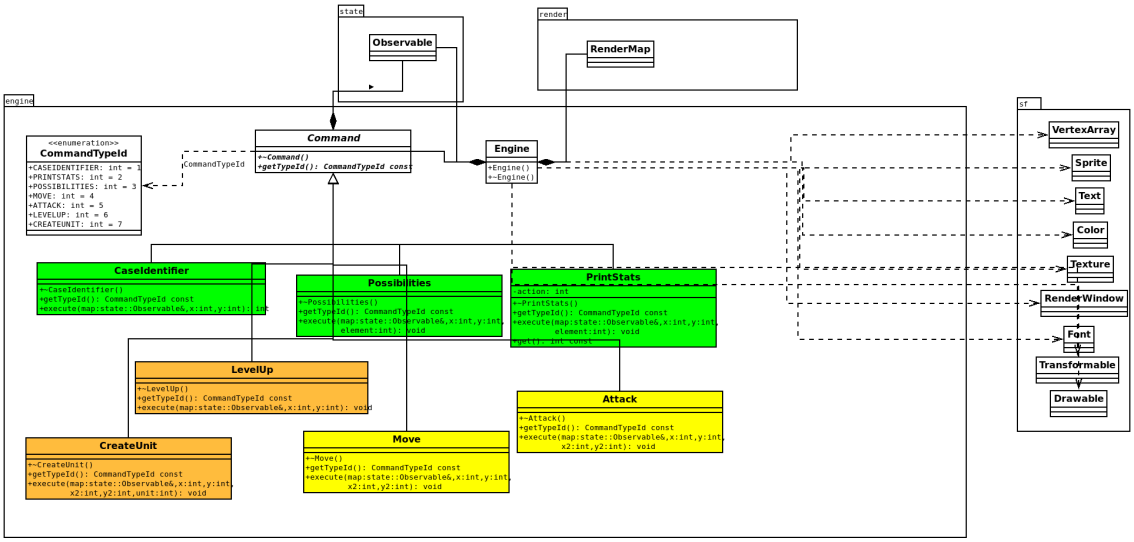

FIGURE 9 – Diagramme de classes du moteur