Rapport projet logiciel transversal

Heroes of might and magic

Mehdi NACER - Mehdi NACER - Maxime HEURTEAU

Table des matières

Objectif

Présentation générale

L'objectif du projet est de réaliser un jeu de stratégie en tour par tour, s'inspirant du jeu « Heroes of might and magic » dont les règles sont développées dans la section suivante.

Règles du jeu

L'univers du jeu est un monde médiéval-fantastique, qui sera représenté sur une carte en 2D avec une vision vue dessus.

Au début du jeu, chaque joueur possède un territoire avec un peuple, un héros et une armée.

Le peuple permet de générer des ressources. Pour se développer, le joueur utilise ces mêmes ressources.

Le jeu peut passer en mode "guerre", lorsque le joueur déclenche une attaque contre une autre armée. L'armée est menée par le héros. Elle suit le héros dans ses déplacements et ses attaques. Si l'armée gagne la guerre, elle gagne des ressources et augmente son niveau, dans le cas contraire elle perd des ressources et diminue son niveau, son armée est aussi vidée de ses soldats. Le principe de la guerre est de vider les points de vie de l'adversaire grâce à des attaques qui auront un niveau donné. Les attaques sont en tour par tour.

Le niveau du jeu sera caractérisé par un chiffre et définie le nombre de soldats par château.

Les différents personnages sont:

- Les villageois: leurs nombres dépendent de la taille du territoire du joueur. Ils permettent de créer des ressources.
- Les soldats: ils existent différents types de soldats: cavaliers, archers... Chacun des soldats doit être entrainés ce qui peut prendre du temps et coûte des ressources. Ils sont contrôlés par le héros et sont définis par des points de vie et une attaque.
- Le héros: le héros est choisi au début du jeu parmi une liste de héros. Il a des pouvoirs spécifiques. Il mène l'armée de soldats pendant les attaques.

Les différents bâtiments sont:

- Les mines: elles contiennent les ressources. Les villageois vont chercher dans ces mines pour avoir des ressources.

- La caserne: elle permet en échange de ressources et d'un temps donné d'avoir des soldats.
- Un château: sert au héros pour se reposer et donnera accès au menu (sauvegarde, langue, son…). Le château aura aussi des points de vies qui seront ajoutés à ceux des soldats lors d'une attaque adverse.

Description et conception des états

Description des états

L'état du jeu est modélisé par deux états : un macro-état pour le jeu en général qui modélise le village et son état par rapport à la map, et un état qui modélise la phase dite de guerre. Chacun des états est formés par un ensemble d'éléments fixes et d'éléments mobiles.

Chaque éléments est défini par :

- Ses coordonnées dans la grille (x,y)
- Sa taille
- Son identifiant d'élément

Etat de jeu village

Cet état modélise le village du joueur, il modélise donc tous les états du village du joueur. Ces éléments fixes sont les bâtiments (château, mine, caserne) et les éléments de décors. Ces éléments mobiles sont les villageois et le héros.

Etats des éléments fixes

La grille est définie en deux types de cases : les cases « libres » et les cases « non-libres ».

Cases libres : Les cases libres sont modélisées par de l'herbe (ou autre élément de décors) qui pourra être chevauché par les éléments mobile.

Cases non-libres : Les cases non-libres ne peuvent être chevauchées par les éléments mobiles et sont soit un bâtiment soit un élément de décors infranchissable (montagne, arbre).

Etats des éléments mobiles

Les différents éléments mobiles sont les villageois et le héros. Les éléments mobiles possèdent une direction (aucune, droite, gauche, haut, bas), une vitesse, une position.

L'élément mobile Héros est défini par :

- Son identité
- Son attaque
- Ses points de vies
- Ses coordonnées sur la map

L'élément mobile villageois est lui définit par :

- Sa coordonnée sur la map

Etat de jeu guerre

Cet état de jeu guerre commence quand un joueur à lancé une guerre contre un second joueur et se finit par la défaite d'un des deux joueurs. L'état de la guerre est donné par l'état des éléments mobiles et fixes mais aussi par l'état dit d'avancé de la guerre.

L'état d'avancé de la guerre est défini par:

- le tour (chaque joueur joue tour à tour
- le temps

Comme pour l'état du jeu village, l'état de jeu guerre est défini par des éléments mobiles et des éléments fixes.

Les éléments fixes sont les éléments de décors qu'ils soient franchissable ou non (comme dans l'état de jeu village).

Les éléments mobiles sont le Héros et son armée (composée des différents types de soldats).

L'élément armée est défini par le nombre de soldats dans l'armée et la somme des points de vies des soldats de l'armée.

Ainsi, chaque soldat est défini par:

- Son type
- Son attaque
- Ses points de vies
- Ses coordonnées sur la map

Le Héros est définis par:

- Son identité
- Son attaque
- Son attaque spéciale
- Ses points de vies
- Ses coordonnées sur la map

État de jeu général

En plus de l'ensemble des éléments vus précédemment, on ajoute les propriétés suivantes:

- L'époque: représente l'heure correspondant à l'état.

- La vitesse: le nombre d'époque par seconde. C'est la vitesse à laquelle l'état est mis à jour.

Conception logiciel

Le diagramme de classe et d'état est à voir en figure n°1.

La classe Element: La Hiérarchie des sous-classes Element permet de définir les différentes catégories et types d'élément. C'est une application du polymorphisme.

La classe Factory: Cette classe va permettre de créer plus facilement les instances d'Element. C'est une application du schéma de conception Abstract Factory.

Ensuite, on utilise des conteneurs d'élément. La classe ElementList contient une liste d'éléments qui sera étendu par la classe ElementGrid qui va permettre de gérer une grille. La classe State est le conteneur principal.

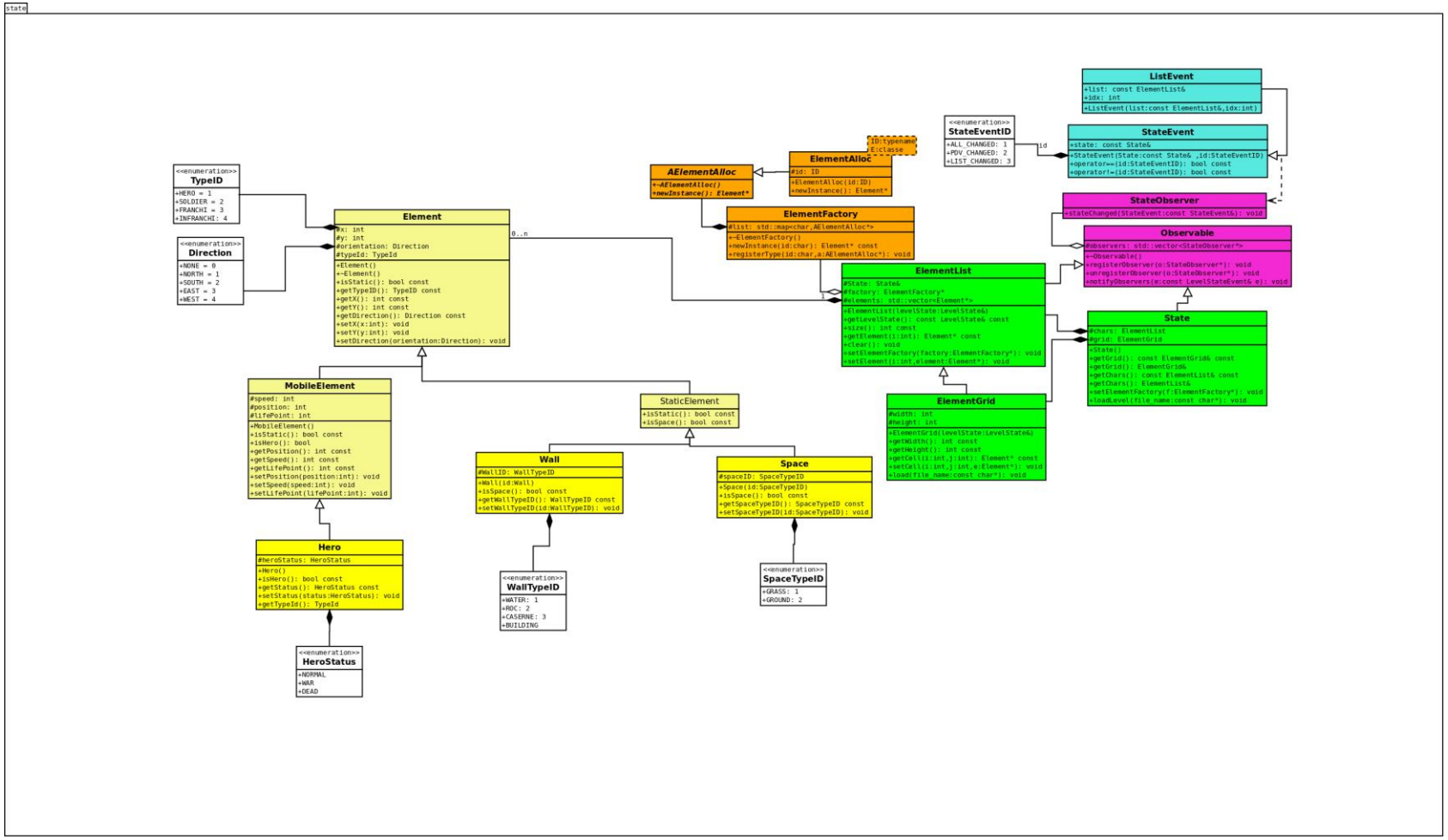

FIGURE 1: DIAGRAMME DES CLASSES

Rendu : Stratégie et Conception

Le but de cette partie est de comprendre comment à partir d'un état de jeu, il est possible d'avoir un rendu graphique de cet état. Pour cela, la librairie sfml est utilisée. Son utilisation permet d'avoir un affichage graphique.

Stratégie d'un rendu d'état

Pour le rendu d'état, nous avons opté pour une stratégie de bas niveau.

Plus précisément, nous découpons la scène à rendre en plans (ou « layers ») : un plan pour le niveau (sol, bâtiment…), un plan pour les éléments mobiles (soldat, héros) et un plan pour les informations (vies, scores, etc.). Chaque plan contiendra deux informations bas-niveau qui seront transmises à la carte graphique : une unique texture contenant les tuiles (ou « sprite »), et une unique matrice avec la position des éléments et les coordonnées dans la texture. En conséquence, chaque plan ne pourra rendre que les éléments dont les tuiles sont présentes dans la texture associée. Pour la formation de ces informations bas-niveau, la première idée est d'observer l'état à rendre, et de réagir lorsqu'un changement se produit. Si le changement dans l'état donne lieu à un changement permanent dans le rendu, on met à jour le morceau de la matrice du plan correspondant. Pour les changements non permanent, comme les animations et/ou les éléments mobiles, nous tiendrons à jour une liste d'éléments visuels à mettre à jour.

Conception logiciel

La conception d'un rendu d'état est visible sur la figure n°2.

L'état du jeu est observé grâce à la classe StateObserver ce qui permet de ne mettre à jour que les éléments qui sont modifié et de le retranscrire dans le rendu. La classe Map va permettre de faire l'affichage à l'écran de tous les éléments. C'est cette classe qui va principalement utiliser la bibliothèque SFML et mettra en place l'interface graphique du jeu.

Ressources

Les ressources sont disponibles sur le site http://spritedatabase.net/ .

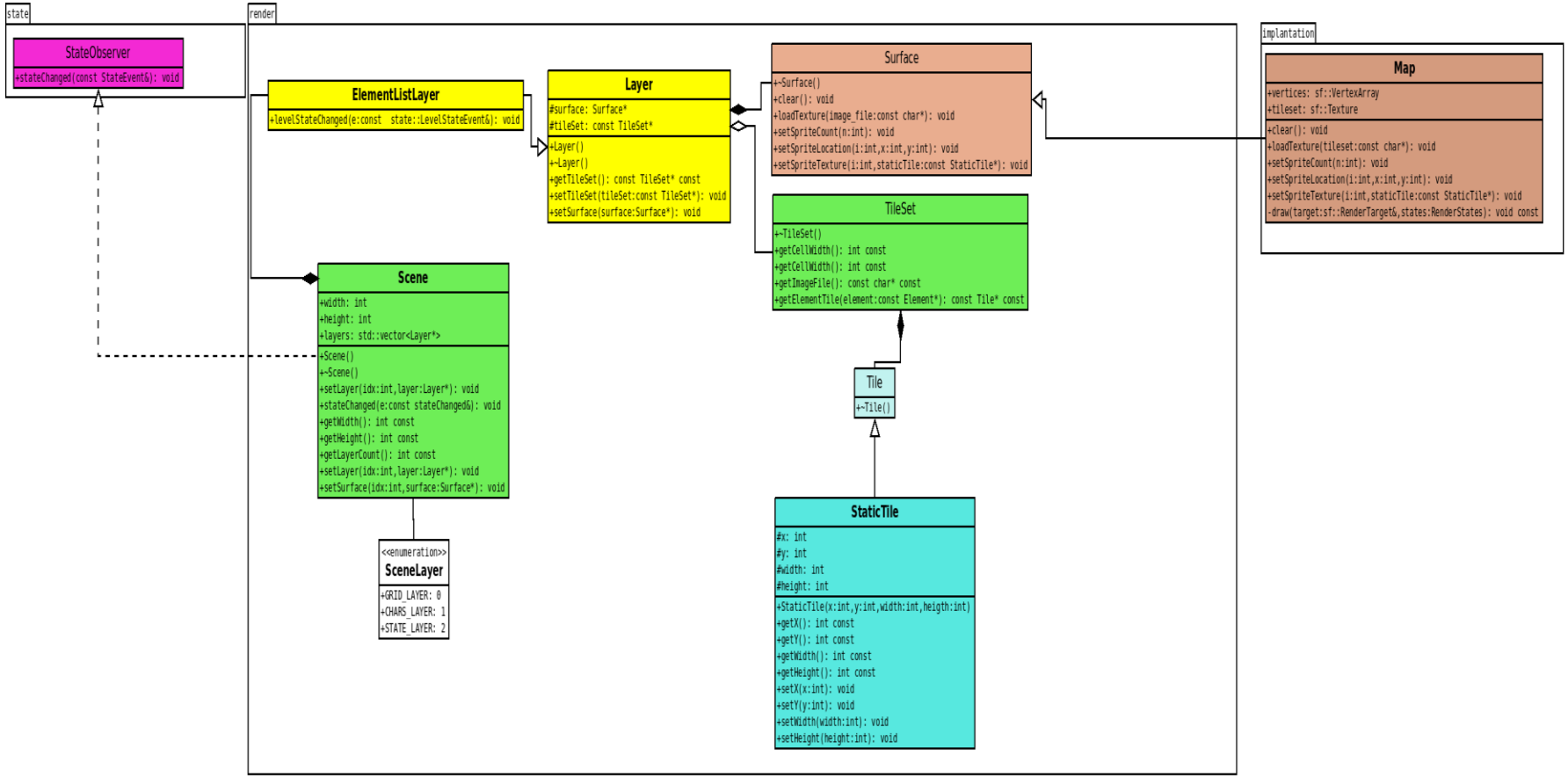

FIGURE 2: DIAGRAMME DES CLASSES POUR LE RENDU

Règle de changement d'états et moteur de jeu

Dans cette partie nous allons détailler les événements qui vont pouvoir changer les états du jeu. Nous allons aussi définir les règles de chronologies de jeu. Ensuite, nous verrons comment nous allons mettre en place tout ceci dans la partie de conception logiciel.

Horloge globale

Une horloge globale est mise en place pour rafraichir les états de jeu. Cette horloge est décorréllée du temps de rafraîchissement de l'affichage ainsi il faut bien choisir ces deux temps pour que le jeu soit fluide.

Changement extérieurs

Les changements extérieurs sont des changements dus à une commande extérieure comme l'appui sur une touche ou un clic.

La première commande mise en place est le déplacement du héros grâce aux flèches du clavier.

Ensuite, on pourra introduire un appui sur un bouton pour faire une attaque.

Enfin on introduira les boutons de commande principale avec la mise en pause, le chargement d'un niveau…

Changement autonomes

Pour les changements autonomes, on vérifie en premier son niveau de vie :

> Si le niveau de vie est nul alors le jeu revient à un état initial.
>
> Sinon le jeu continu et passe à la suite.

On vérifie les attaques sur le héros :

> Si il subit une attaque, on soustrait c'est points de vies de la valeur de l'attaque
>
> Sinon il ne se passe rien.

Ces changements sont une visualisation simple de ce qui sera produit dans le futur, et seront améliorer dans les étapes suivantes.

Conception logiciel

Le diagramme des classes est visible sur la figure n°2. L'ensemble du moteur de jeu est mis en place sur le patron de type command.

Le diagramme est fait pour une seule action de changements d'états de jeu fait par le moteur de jeu mais permet de voir la philosophie du moteur.

Ainsi plusieurs classes sont dominantes tels que :

La classe **Engine :** elle est le cœur du moteur du jeu, elle stocke les commandes dans une instance de CommanSet. Après un temps suffisant on appelle la méthode update pour transférer les commandes.

Les classes **Action :** Elles permettent de mettre en œuvre une commande dans l'état de jeu c'est-à-dire modifier l'état de jeu. Par exemple, la méthode DirectionCommand va modifier la direction de l'état de jeu.

La classe **Command :** Cette classe permet de gérer les commandes extérieures en fonction des interactions extérieures (gérer par le moteur de rendu)

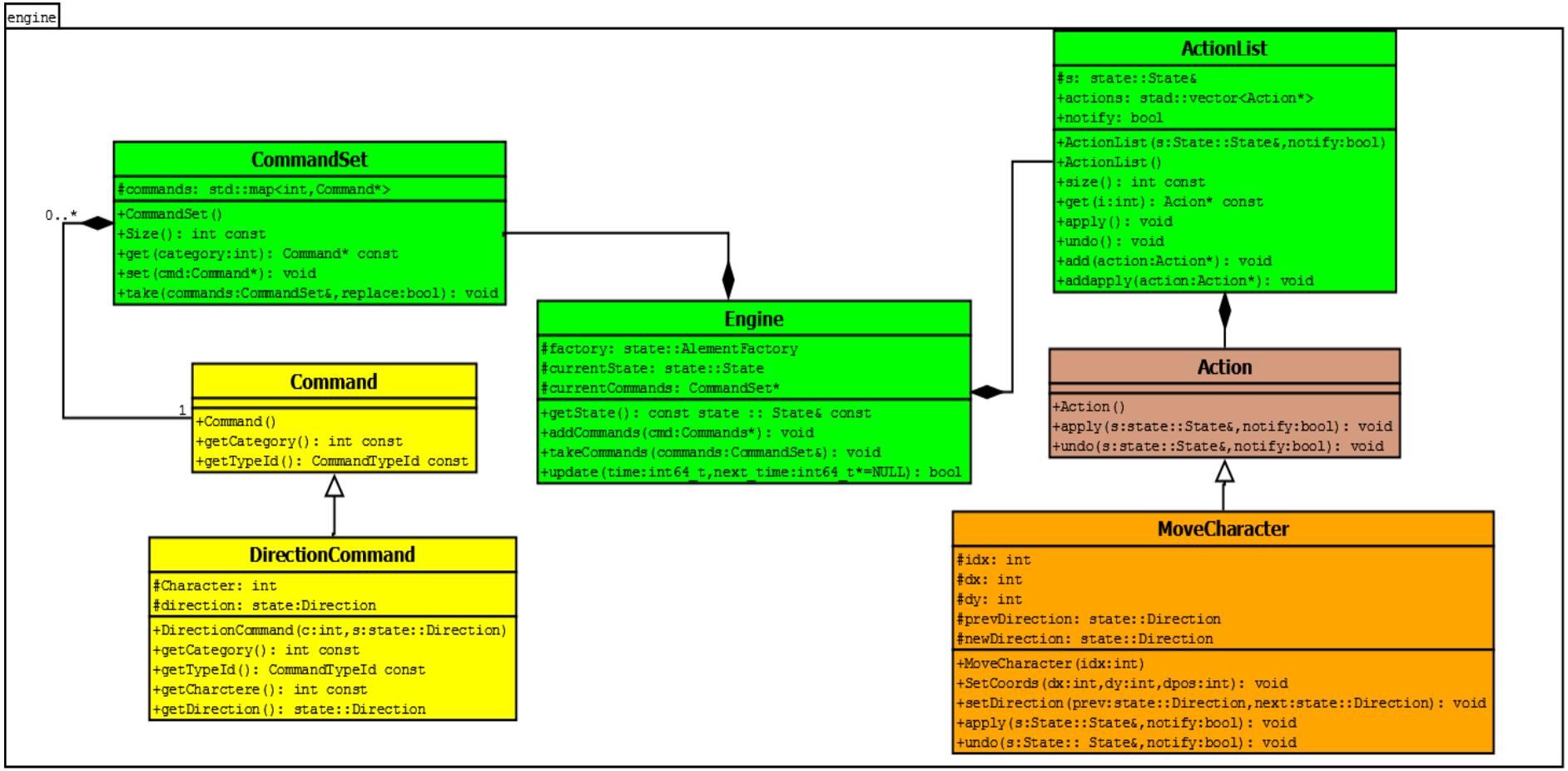

FIGURE 3: DIAGRAMME DES CLASSES POUR LE MOTEUR DE JEU

Intelligence artificielle

Dans cette partie nous allons développer la conception de l'intelligence artificielle, c'est ici que nous allons définir les interactions que doit faire cette intelligence pour qu'elle puisse jouer au jeu comme si nous avions un joueur.

Intelligence artificiel simple

Dans le but d'avoir une sorte d'étalon, mais également un comportement par défaut lorsqu'il n'y a pas de critère pour choisir un comportement, nous proposons une intelligence extrêmement simple, basée sur les principes suivants :

Tant que c'est possible on avance : on ne fait demi-tour que si on est dans un cul de sac. Cela supprime les mouvements erratiques.

Lorsqu'on arrive à une bifurcation, on choisit aléatoirement l'une des possibilités. Cela permet d'assurer l'exploration complète du niveau, tout en rendant les mouvements imprévisibles.

Nous allons ensuite proposer une suite d'heuristiques pour avoir un comportement qui se rapproche le plus d'un comportement joueur humain.

Un héros lorsqu'il se déplace se rapproche de plus en plus de ses ennemis et lorsqu'il se trouve à une certaine distance (assez proche) alors il attaque.

Pour implémenter ceci nous allons introduire un calcul de distance entre héros et ennemis.

Modularisation

Dans cette partie nous allons développer la répartition des différents modules de jeu dans des processus différents. La répartition des modules sera faite dans différents threads.

Deux threads sont utilisés une pour le moteur de jeu et une pour le moteur de rendu.