Projet “Garden Tensions”

Agneray Aurore
Lafage Victoire

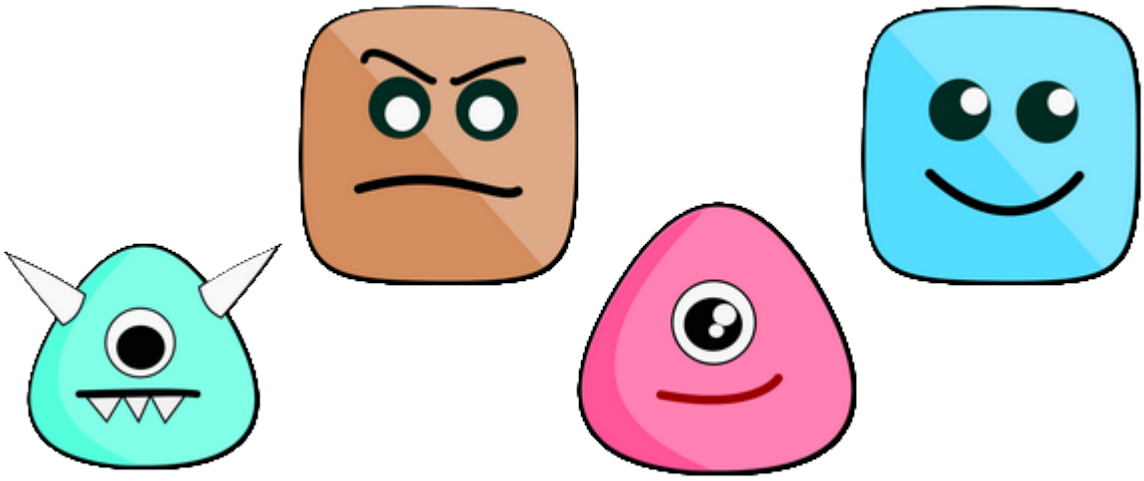

FIGURE 1 - Les “Bloubii”, petites créatures fort curieuses !

Table des matières

1. Présentation générale

1.1. Prologue

Durant une nuit de pleine lune, les Bloubii atterrirent par hasard sur Terre, et plus particulièrement dans ce que nous appelons communément un “jardin”, lors d’un test de leur portail spatio-temporel. Mais une fois le soleil levé, étant minuscules, cet espace leur sembla empli de richesses et curiosités infinies à découvrir. Les créatures se dispersèrent alors pour explorer l’endroit et le trouvèrent bien attrayant. De l’herbe, du sable et de la boue à perte de vue ...

Toutefois des zones incongrues leur furent inaccessibles. Nous les nommons le “ciel”, les “sucreries” (enfermées dans un bocal), le “barbecue” et la “piscine”. Ils ne savent en effet ni voler, ni ouvrir un bocal 1000 fois plus haut qu’eux, ni supporter la chaleur, ni nager. Pourtant leur curiosité intarissable va les motiver à chercher des solutions. Malheureusement, n’en trouvant pas rapidement, des tensions vont naître dans leurs rangs, et des clans vont se former ... Leur objectif ? Conquérir tout le jardin, ainsi que le ciel ! Pour un jour, peut-être, explorer d’autres contrées lointaines ...

1.2. Archétype

Ce projet est globalement inspiré du jeu Risk du point de vue des règles de conquêtes de territoires. En ce qui concerne la gestion des ressources et des cellules, cela s’inspire plutôt de Civilization.

1.3. Règles du jeu

Le jeu se déroule sur un plan de base divisé en un nombre fixé de 29 cellules hexagonales. La majorité des cellules se regroupent en zones (zone sable, zone herbe, zone boue). Néanmoins quelques cellules sont spéciales et offrent des avantages une fois qu’elles sont conquises. Il s’agit du ciel, du pot de sucreries, du barbecue, et de la piscine. En contrepartie, elles sont aussi plus difficiles à conquérir, dans la mesure où il faut avoir accumulé un certain nombre de ressources pour pouvoir y accéder. (Cf figure 1 ci-dessous) Le plan est généré aléatoirement et change à chaque partie.

Chaque joueur dispose en début de partie d’une seule cellule ainsi que de trois créatures, dont deux permettent de conquérir des cellules adjacentes. En effet sur toute cellule conquise doit rester au moins une créature.

Type de cellule	Technologie à développer	Ressources nécessaires	Avantage apporté (nécessite de dépenser les ressources listées à gauche à chaque utilisation)
Herbe / Sable / Boue	Aucune	Aucune	Apport de ressources diverses **A CHAQUE TOUR**
Ciel	Jet-pack	5 métaux	Attaque possible de n'importe quelle case adverse
Sucreries	Super-protéines (pour avoir des supers muscles)	3 nourritures	Génération de 3 personnages supplémentaires
Barbecue	Armure de pierre	3 pierres	Possibilité de tuer gratuitement 3 personnages d'une case adverse
Piscine	Radeau	3 bois	Idem

FIGURE 2 - Ressources nécessaires pour conquérir les cellules spéciales et avantages qu'elles procurent une fois conquises

Avant le lancement de la partie, le joueur doit choisir le clan de créatures à contrôler. Il en existe 4 ayant chacun ses propres caractéristiques : les cuisiniers, les forgerons, les mineurs et les bûcherons. Lors des phases de conquête, chaque combat remporté génère un point de conquête, et lorsque le joueur cumule 8 points de conquête, il peut ordonner à l'une de ses créatures d'effectuer une capacité spéciale, soit un pillage qui permet de voler des ressources en plus d'attaquer, soit une attaque de zone qui affecte la cellule visée par la capacité. (Cf figure 2 ci-dessous pour le détail de ces capacités)

Capacité spéciale	Cuisiniers	Forgerons	Mineurs	Bûcherons
Pillage	Vole 2 quantités de **nourriture** à l'adversaire et tue une créature présente sur la cellule	Vole 2 quantités de **métaux** à l'adversaire et tue une créature présente sur la cellule	Vole 2 quantités de **pierres** à l'adversaire et tue une créature présente sur la cellule	Vole 2 quantités de **bois** à l'adversaire et tue une créature présente sur la cellule
Attaque de zone	Empoisonnement de la cellule visée pendant 3 tours (à chaque tour, chaque créature présente sur la cellule tire un nombre au hasard entre 1 et 6, un autre nombre est tiré par le système, si celui de la créature est plus faible, elle meurt), peut affecter tous les joueurs.	Les créatures utilisent les armes qu'elles ont forgées et infligent ainsi plus de dégâts jusqu'à la fin du prochain tour (leur dégâts varient de 1 à 9 au lieu de 1 à 6).	Les créatures lancent des rochers sur les adversaires de la cellule visée, ce qui immobilise tous les adversaires, ils ne peuvent alors pas conquérir de territoire pendant 2 tours.	Les créatures conquièrent la cellule visée en construisant un abri (qui immunise contre une frappe). Les créatures adverses initialement présentes sur la cellule sont déportées, elle disparaissent temporairement du plan de jeu pendant ce tour et réapparaissent pendant la phase de renfort de l'adversaire qui doit alors les replacer.

FIGURE 3 - Capacités spéciales des clans

Le jeu se déroule au tour par tour, chaque tour étant composé de 2 phases : la phase de conquête et la phase de renfort.

<u>Phase de conquête :</u>

Durant cette phase, le joueur peut conquérir des cellules adjacentes à ses propres cellules. Celles-ci peuvent être vides ou déjà conquises par son adversaire. Dans ce dernier cas, un combat a lieu.

Déroulement d'un combat :

Chaque joueur tire un nombre entre 1 et 6 autant de fois qu'il a de créatures sur sa cellule concernée par le combat, le score du joueur est la somme de ces nombres. Le joueur qui a le score le plus élevé remporte le combat.

- Si l'attaquant remporte le combat avec N créatures, la cellule du défenseur lui appartient et elle dispose alors de N - 1 créatures, la dernière étant restée sur la cellule de l'attaquant.
- Si le défenseur remporte le combat, sa cellule reste intacte et la cellule de l'assaillant ayant attaqué avec N créatures n'en dispose désormais que d'une seule.

- S’il y a égalité, on considère que le défenseur a remporté le combat et la règle précédente s’applique.

Phase de renfort :

Durant cette phase le joueur gagne un certain nombre de créatures, il en gagne automatiquement M ainsi que N supplémentaires par zone sous contrôle (une zone est sous contrôle si toutes les cellules du type de cette zone appartiennent au joueur). De plus si le joueur a conquis la cellule spéciale Sucreries, il gagne K créatures supplémentaires.
Il doit ensuite disposer l’ensemble de ces nouvelles unités sur les cellules de son territoire sachant que chaque cellule ne peut pas accueillir plus de 5 créatures.

Seulement si le temps le permet : Les humains interviennent de façon aléatoire au cours du jeu, ils peuvent par exemple marcher sur une cellule et éventuellement tuer des créatures qui s’y trouvent, utiliser le barbecue ou la piscine, tuant toutes les créatures présentes sur ces cellules spéciales, des enfants humains peuvent également faire tomber des sucreries dans le jardin, permettant aux créatures de se nourrir et ainsi de se multiplier plus facilement.

1.4. Ressources

Globalement nous aurons besoin :

- de cellules hexagonales (image format png 512*512 pixels) :

FIGURE 4 - Hexagones représentant les cases

- d’un filtre hexagonal coloré pour signaler l’appartenance d’une cellule à un joueur
- d’un fond d’écran (pour le moment on considère que le fond sera uni)
- des groupes de personnages (image format png 512*512 pixels) :

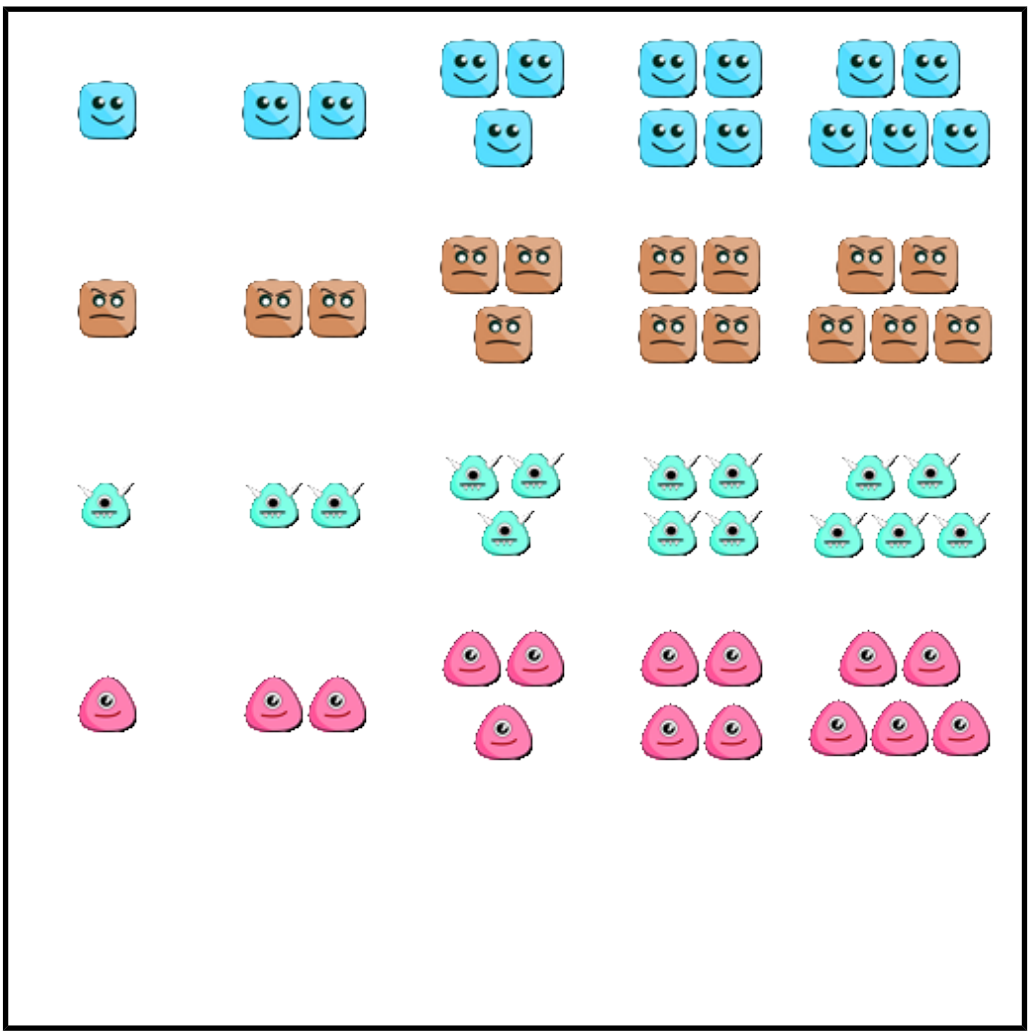

FIGURE 5 - Textures pour les personnages

2. Description et conception des états

2.1. Description des états

Un état du jeu est composé par des éléments qui seront dans une grille hexagonale, et d'autres qui seront hors de la grille. Dans la grille, il y aura des éléments fixes (les cellules hexagonales), ainsi que des éléments déplaçables par le joueur : les créatures ainsi que le filtre indiquant quelle case est sélectionnée.

Dans notre code nous créerons donc **deux tableaux d'éléments** : un contenant les cellules hexagonales, et un autre contenant les créatures disposées sur la grille de jeu.

Tous les éléments de la grille doivent posséder les propriétés suivantes :

- Coordonnées (x,y) dans la grille
- Identifiant de type d'élément (nombre qui indiquera la classe de l'élément)
- Le nombre de créatures présentes (soit sur la cellule, soit dans le groupe de créatures)

Il faudra aussi gérer l'état de chaque joueur en lice, **en dehors des tableaux et donc de la grille**. Il s'agira de garder les propriétés :

- Nom du clan
- Liste des créatures appartenant au joueur disposées sur la carte
- Nombre de créatures en stock (qui n'ont pas encore été disposées sur la carte lors de la phase de renfort)
- Noms des cellules spéciales conquises
- Nombre de points de conquête
- Coordonnées de la dernière cellule sélectionnée
- Quantités de ressources possédées (nourriture, métaux, pierres, bois)
- Nombre de zones (herbe, boue ou sable) conquises

2.1.1. Etat des éléments fixes

Un premier tableau comportera des éléments nommés «cellules». Sa taille est fixée au démarrage du niveau. Toutes les cellules seront franchissables par les éléments déplaçables.

Elles devront donc comporter les propriétés :

- Nombre de personnages présents sur la cellule (attribut issu de la classe mère)
- Joueur présent sur la cellule (désignera la couleur du filtre de la cellule)
- Etat de la cellule lié aux capacités utilisées par les joueurs : normal, empoisonnée ou protégée

Nous travaillerons néanmoins sur plusieurs types de cellules, qui auront chacun leur spécificité.

Les trois types de cellules qui suivent sont des cellules de zones qui génèrent des ressources.

- Les cellules « herbe » : comporteront la texture herbe
- Les cellules «sable» : comporteront la texture sable

- Les cellules «boue» : comporteront la texture boue

Les quatre types de cellules suivants apportent un pouvoir au clan auquel elles appartiennent. On associera ce pouvoir dans le moteur de jeu grâce à l'attribut SpecialCellID.

- La cellule «ciel» : comportera la texture ciel/jet-pack. Permet d'attaquer n'importe quelle case adverse, quelle que soit sa position.
- La cellule «sucreries» : comportera la texture sucreries. Permet de générer trois personnages supplémentaires lors de la phase de renfort.
- La cellule «barbecue» : comportera la texture barbecue. Permet de tuer gratuitement 3 personnages d'une case adverse adjacente.
- La cellule «piscine» : comportera la texture piscine. Même pouvoir que pour la case «barbecue ».

2.1.2. Etat des éléments déplaçables

Les éléments déplaçables seront déplacés comme des pions par le joueur. Ceux présents sur la grille seront stockés dans un tableau dans la classe State, tandis que d'autres seront en possession du joueur. A chaque élément doit donc être associé le joueur auquel il appartient.

Élément déplaçable « Ensemble de personnages » : Lorsqu'un joueur attaque une cellule adverse, c'est l'ensemble des personnages d'une de ses cellules qui part au combat. On doit donc disposer d'une classe qui sera un groupe de personnages. En outre lors de la phase de renfort, le joueur doit pouvoir disposer individuellement ses personnages sur une case. C'est pourquoi un groupe de personnages peut comporter entre 1 et 5 personnages. Cette classe doit avoir comme propriétés le nombre de personnages concernés.

2.1.3. Etat général

A un niveau supérieur aux éléments décrits précédemment, nous devons ajouter :

- Le nombre total de cellules de la grille
- Un tableau d'éléments "Cellules"
- Un tableau d'éléments "Groupe de créatures"

2.2. Conception Logicielle

Nous avons choisi de mettre en évidence les classes suivantes dans notre diagramme des classes :

Nom	Rôle de la classe
State	Représente l'état actuel du jeu : contient un tableau des créatures présentes sur la grille de jeu et un tableau contenant les cellules de la grille, ainsi que le nombre de cellules de la grille.
ElementTab	C'est un tableau composé d'éléments. Nous lui avons attribué deux classes filles afin que l'une d'elle s'occupe des créatures et l'autre des cellules.
Element	Peut être une cellule ou un groupe de créatures, permet d'obtenir leur position
Player	Regroupe l'ensemble des variables dl'état du joueur
Cell	Elément fixe de la grille
<<enumeration>> CellState	Identifie l'état de la cellule, ce qui permettra plus tard de modifier les phases de combat ayant lieu sur celle-ci.
SimpleCell	Type de cellule destiné à fournir des ressources au joueur. La classe identifie le nombre et le type de ressources que la cellule offre à chaque tour.
SpecialCell	Type de cellule destiné à fournir des pouvoirs supplémentaires au joueur. Identifie le nombre et le type de ressources dont la cellule a besoin pour déclencher son pouvoir.
<<enumeration>> ID	Caractérise le type d'élément, que ce soit une cellule ou un groupe de créatures : sable, boue ou herbe
<<enumeration>> TypeID	Caractérise le type d'élément, généralise l'ensemble des éléments de l'énumération précédente.
CreaturesGroup	Elements déplaçables du jeu.
<<enumeration>> CreaturesID	Identifie le type de créatures. Permettra plus tard de caractériser leur type d'attaque ainsi que les ressources qu'elle peuvent voler
PlayerRessources	Donne l'état des ressources du joueur, peut également les modifier
<<enumeration>> ClanNameID	Associe à chaque joueur un type de créatures

Voici notre diagramme des classes de l'état :

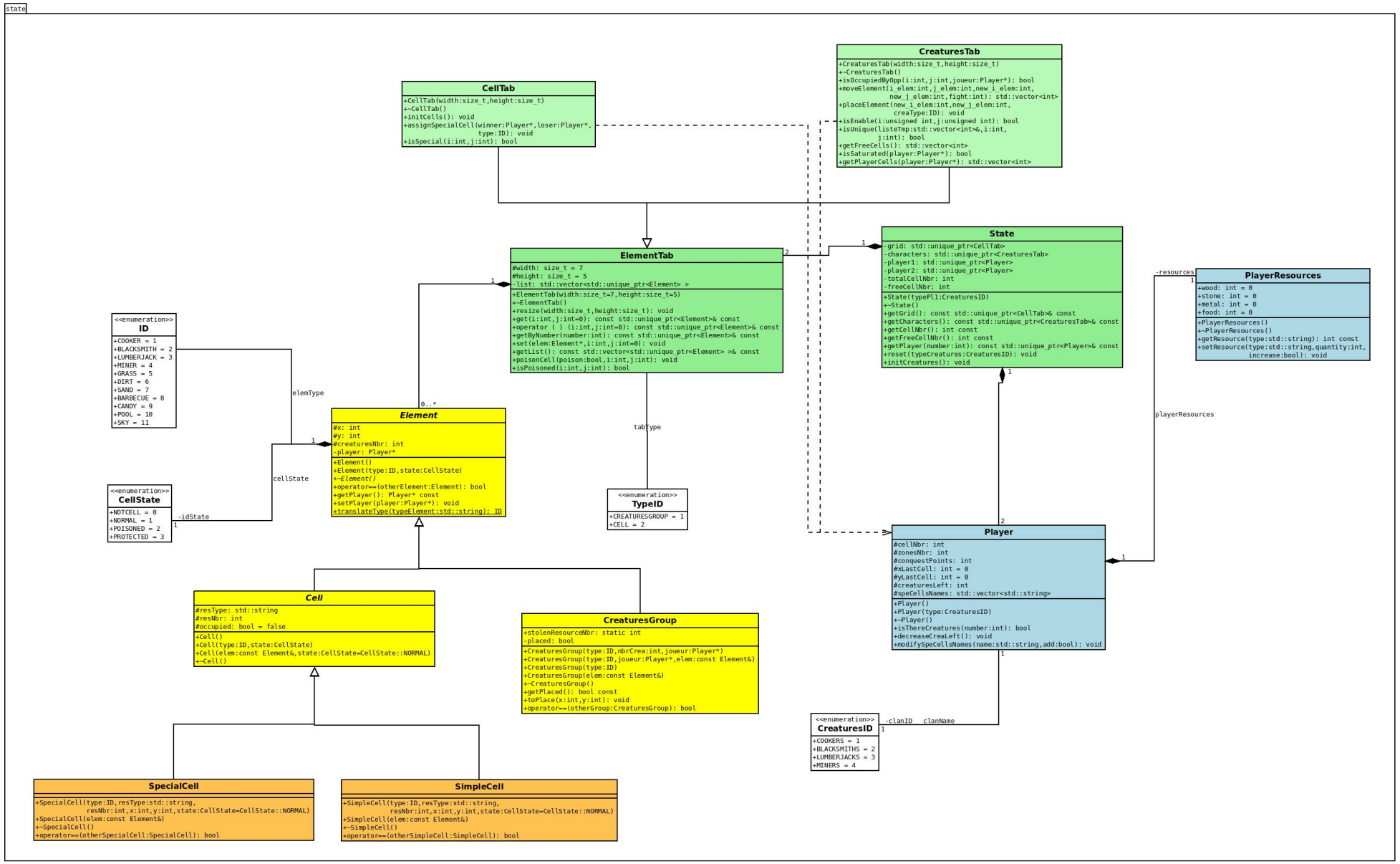

FIGURE 6 – Diagramme des classes de l’état

3. Rendu : stratégie et conception

3.1. Stratégie de rendu d'un état

Pour le rendu d'un état, nous découpons la scène à rendre en trois plans :

- le plan contenant les cellules
- le plan contenant les créatures disposées sur la carte
- le plan des informations (nombre de créatures que l'on peut placer, pouvoirs disponibles, etc)

Chaque plan contiendra deux informations qui seront transmises à la carte graphique : une unique texture contenant les cellules hexagonales et une unique texture avec la position des éléments et les coordonnées dans la texture.

Le plan comportant les cellules sera initialisé en début de partie et une fois la partie commencée les cellules gardent la même place. La seule modification qui leur sera apportée sera **éventuellement** d'imposer un contour ou un filtre coloré correspondant au joueur 1 ou au joueur 2 selon l'appartenance de la cellule à l'un d'entre eux. Ces changements seront gérés en mettant à jour la matrice du plan.

Nous avons pour cela besoin d'une horloge qui aura comme ordre de grandeur de fréquence 4-12 Hz.

3.2. Conception logiciel

Voilà notre diagramme des classes de rendu :

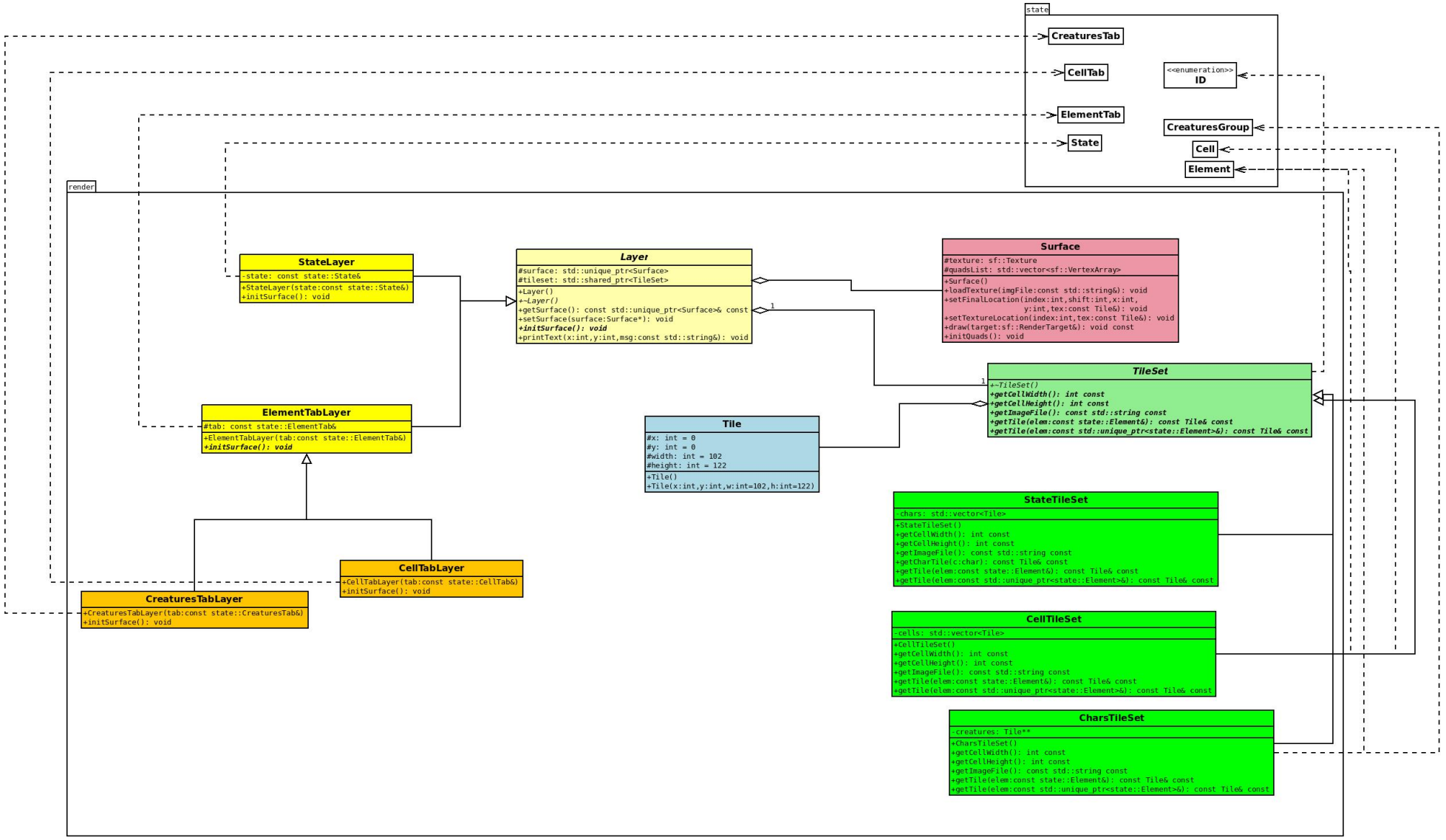

FIGURE 7 – Diagramme des classes du rendu

4. Règles de changement d’état et moteur de jeu

4.1. Règles de changement d’état

L’état change à chaque action d’un joueur. A chaque tour, le joueur détient un certain nombre de ressources, de cellules, de créatures à placer, et de points de conquête.

Au début du tour, le nombre de ressources du joueur augmente automatiquement en fonction des cellules conquises.

Lorsque le joueur conquiert une cellule simple, son nombre de cellules possédées augmente, l’état des cellules correspondantes est mis à jour, tout comme l’état des groupes de créatures.

Lorsque le joueur conquiert une cellule spéciale, son nombre de cellules possédées augmente, et celui de ses ressources diminue en fonction des ressources nécessaires. L’état des cellules correspondantes est mis à jour, tout comme l’état des groupes de créatures.

Si le joueur active le pouvoir d’une cellule spéciale, l’état de ses ressources est mis à jour, ainsi que l’état de la cellule et du groupe de créatures affectés par le pouvoir.

Si le joueur active la capacité de ses créatures, son nombre de points de conquête change, si on utilise une capacité de pillage, le nombre de ressources change également, de plus l’état des groupes de créatures est modifié.

A la fin du tour, le joueur place les créatures non placées qu’il détient sur les cellules qu’il possède, ce qui met à jour l’état des groupes de créatures ainsi que son nombre de créatures restantes à placer.

4.2. Changements en bref

Ils sont provoqués par des commandes sélectionnées par le joueur. Il s’agira globalement ici de clics sur des cases avec la souris.

- Commandes principales :
    - Démarrer une nouvelle partie
        - Choix d’un clan de créatures par le joueur
    - Ajouter un groupe de créatures sur la carte
    - Déplacer un groupe de créatures déjà présent sur la carte afin de conquérir une cellule (vide ou détenue par l’adversaire), en deux étapes :
        - Sélection de la cellule où se trouve le groupe de créatures
        - Sélection de la cellule où l’on souhaite placer ce groupe
    - Utiliser la capacité spéciale du clan (possible si on dispose de 8 points de conquête à dépenser). Doit amener à sélectionner la cellule adverse qu’on souhaite attaquer.

- Utiliser une capacité liée à l'une des quatre cellules spéciales (possible si on dispose du bon nombre de ressources à dépenser, cf figure 2). Doit amener à sélectionner la cellule adverse qu'on souhaite attaquer.

4.3. Changements détaillés

- Au début de chaque tour, le joueur gagne automatiquement les ressources liées aux cellules simples (boue / herbe / sable) qu'il possède.

🔷 **Phase de conquête :** durant cette première phase, le joueur peut conquérir des cellules vides ou contenant des groupes des créatures adverses.

- Si le joueur sélectionne une cellule où se trouve l'un de ses groupes de créatures, elle s'éclaircit (passe en mode sélectionnée).

- Après avoir sélectionné une cellule où se trouve un groupe, si le joueur sélectionne une autre cellule :
  - la cellule est vide : on déplace le groupe de créatures. Le groupe est scindé : une créature reste dans la cellule de départ tandis que les autres se déplacent dans la nouvelle cellule.
  - la cellule contient un groupe du joueur : il ne se passe rien
  - la cellule contient un groupe adverse : on engage un combat. Le groupe est scindé : une créature reste dans la cellule de départ tandis que les autres se déplacent dans la nouvelle cellule pour mener le combat.

- Comment se passe un combat ?
  Pour chaque joueur il y a tirage au sort de nombres entre 1 et 6 autant de fois qu'ils disposent de créatures engagées dans le combat. Leur score correspondra alors à la somme de leurs lancers de dés. Le joueur qui remporte le combat (et donc la cellule) est celui dont le score est le plus élevé. De plus le gagnant remporte 1 point de conquête.

- Conquête d'une cellule simple : le nombre de cellules possédées augmente, l'état des cellules correspondantes est mis à jour, tout comme l'état des groupes de créatures.

- Conquête d'une cellule spéciale : le nombre de cellules possédées augmente, et celui des ressources diminue en fonction des ressources nécessaires. L'état des cellules correspondantes est mis à jour, tout comme l'état des groupes de créatures.

- Utilisation de la capacité spéciale du clan sur une cellule adverse : dépense de huit points de conquête.

Capacité spéciale	Cuisiniers	Forgerons	Mineurs	Bûcherons
Pillage	Pour l'adversaire : -2 de **nourriture** et -1 créature sur la cellule	Pour l'adversaire : -2 de **métaux** et -1 créature sur la cellule	Pour l'adversaire : -2 de **pierre** et -1 créature sur la cellule	Pour l'adversaire : -2 de **bois** et -1 créature sur la cellule
Attaque de cellule	L'état de la cellule passe à POISONED pendant 3 tours, peut affecter tous les joueurs.	Les créatures présentes gagnent un bonus de dégâts jusqu'à la fin du tour suivant. (de 1 à 9 au lieu de 1 à 6)	Les créatures adverses présentes sur la cellule sont immobilisées pendant 2 tours. (ne peuvent plus conquérir)	Le joueur conquit la cellule visée en l'immunisant contre une frappe. (passe à l'état PROTECTED) Les créatures adverses initialement présentes sur la cellule sont ajoutées à son compteur de créatures à placer lors de sa prochaine phase de renfort.

Etat POISONED : pendant trois tours, chaque créature présente sur la cellule tire un nombre au hasard entre 1 et 6, un autre nombre est tiré par le jeu. Si celui de la créature est le plus faible, elle meurt.

- Utilisation d'une capacité liée à une cellule spéciale : dépense du type et du nombre de ressources nécessaires.

Type de cellule spéciale	Ressources nécessaires	Avantage apporté
Ciel	5 métaux	Attaque possible de n'importe quelle case adverse
Sucreries	3 nourritures	Génération de 3 personnages supplémentaires lors de la phase de renfort
Barbecue	3 pierres	Possibilité de tuer gratuitement 3 personnages d'une case adverse
Piscine	3 bois	Idem

🔷 **Phase de renfort :** durant cette seconde phase, le joueur gagne un certain nombre de créatures (au minimum M) fonction du nombre de zones de cellules qu'il possède ainsi que du fait qu'il possède ou non la cellule « Sucreries ».

- Si le joueur ajoute un groupe de créatures sur la carte, son nombre de groupes disponibles est décrémenté de 1.

4.4. Conception logicielle

Le moteur de jeu reposera sur l'utilisation d'un patron de conception de type Command, et a pour intérêt de permettre au joueur d'effectuer des commandes extérieures qui modifieront l'état du jeu.

Classes Command : Le rôle de ces classes est de représenter une commande, quelque soit sa source (ici ce sera la souris). En effet ces classes ne gèrent pas l'origine des commandes, dans la mesure où leurs instances seront fabriquées en dehors du moteur de jeu. A ces classes on associe un type de commande avec CommandeTypeID pour identifier précédemment la classe d'une instance. Elles testent la possibilité d'effectuer une action puis si celle ci est possible l'applique.

- MoveCharCommand : vérifie que la cellule visée se trouve parmi les cellules disponibles
- FightCommand : Teste si un combat doit avoir lieu
- PoisonCommand : Teste si la cellule concernée est POISONED et appartient au joueur adverse
- ProtectedCommand : Vérifie que la cellule visée appartient bien au joueur et qu'elle n'est pas déjà protégée

Classes Action : Le rôle de ces classe est d'appliquer les actions que représentent chaque commande précédemment citée.

- NewGameAction : génère un nouveau jeu
- MoveCharAction : déplace un groupe de créatures suivant une cellule de départ et une cellule d'arrivée, indiquées par le joueur
- FightAction : En fonction de l'issue du combat, effectue les changements nécessaires
- PoisonAction : Tue potentiellement les créatures présentes sur la case visée.
- ProtectedActon : Apporte les modifications nécessaires lorsqu'une cellule passe à l'état PROTECTED

Engine : C'est la classe centrale du moteur. Elle stocke les commandes dans une std::map avec une clé entière. On peut par ce procédé définir des priorités : on traite les commandes dans l'ordre de leurs clés, de la plus petite à la plus grande. Lorsque le joueur effectue une action (appelle une commande donc), le moteur appelle la méthode execute() de chaque commande et supprime toutes les commandes.

Voici notre diagramme des classes pour le moteur :

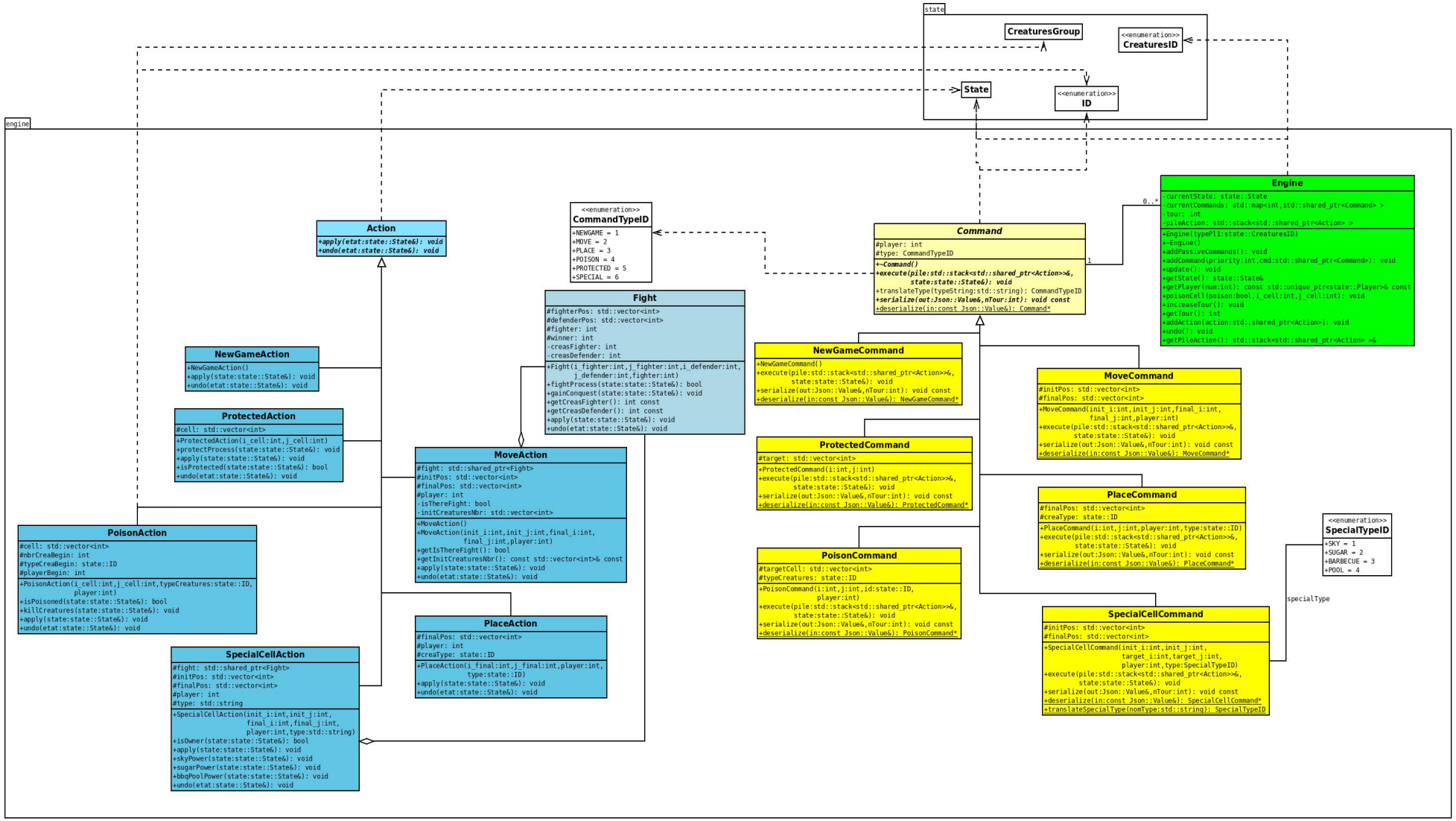

FIGURE 8 – Diagramme des classes du moteur

5. Intelligence artificielle

Nous avons pour le moment réussi à mettre en place la possibilité pour les créatures de se déplacer. Il y a alors plusieurs cas pour la cellule de destination :

- elle est vide
- elle est occupée par le joueur dont c'est le tour
- elle est occupée par l'adversaire

5.1. Intelligence aléatoire

L'IA aléatoire que nous avions mis en place ajoute dans sa liste de commandes 5 commandes de déplacement de créatures déjà existantes sur le plateau et 5 commandes d'ajout de créatures en renfort (qui étaient donc hors plateau). Elle sélectionne ces 10 commandes selon les possibilités que lui offre l'état du jeu.

Ensuite, elle en tire 5 au hasard parmi les 10 et les exécute.

5.2. Intelligence heuristique

Nous avons choisi de faire en sorte que cette IA mette la priorité sur les déplacements vers les cellules adverses dans l'optique de le combattre. Le choix d'un déplacement de créatures de l'IA se fait donc ainsi :

- on détecte les cellules de l'IA qui ont des cellules adverses qui leur sont adjacentes
- parmi ces cellules de l'IA, on récupère les coordonnées de celle qui contient le plus de créatures (on l'appelle par exemple A)
- enfin, parmi les cellules adverses adjacentes de A, on récupère les coordonnées de celles qui comporte le moins de créatures
- on envoie alors la commande de déplacement dans la liste des commandes

Dans le cas où l'IA n'est adjacente à aucune cellule de son adversaire, elle effectuera des déplacements aléatoires de ses créatures vers des cases vides.

5.3. Intelligence avancée

Cette IA se base sur un arbre binaire des états, pour cela nous avons implémenté pour chaque classe action une méthode undo permettant de revenir d'un état à son état précédent afin de parcourir l'arbre aisément et de calculer les différents poids attribués à chaque état afin de déterminer le meilleur chemin possible grâce à la fonction minimax.

Nous avons en outre commencé à mettre en place des classes de path finding afin de permettre à l'IA de se rapprocher de son adversaire afin de conquérir ses territoires. On utilisera notamment l'algorithme de dijkstra afin de déterminer le chemin le plus court vers l'adversaire. Les classes relatives à cela sont en cours de tests.

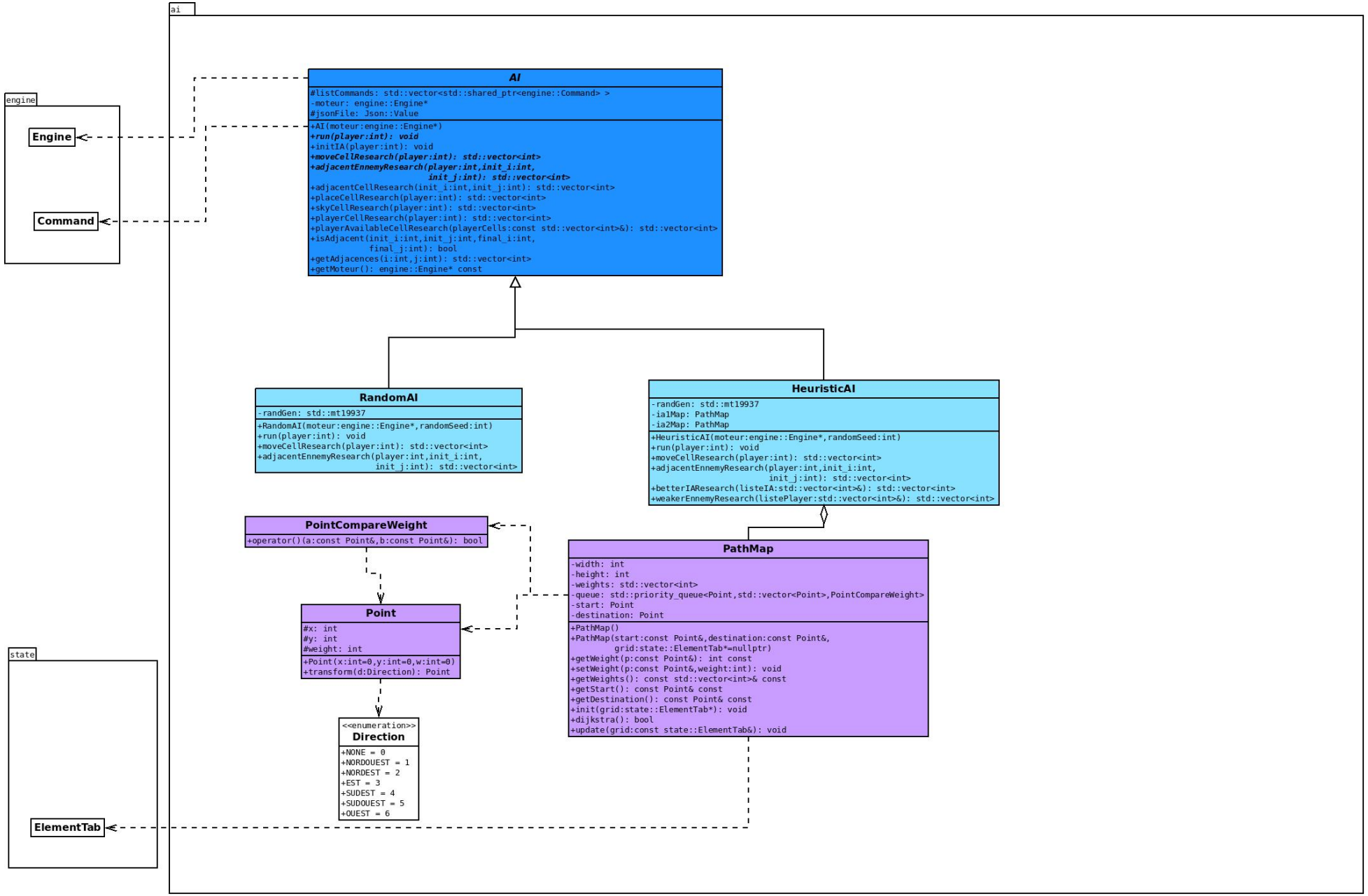

FIGURE 9 – Diagramme des classes de l’intelligence artificielle

6. Modularisation

En ce qui concerne l'exécution de ce dernier livrable, côté client tout se passe dans le fichier TestsNetwork.cpp.

6.1. Organisation des modules

6.1.1. Répartition sur différents threads

Le fichier TestsNetwork.cpp comporte une méthode nouvellePartie() qui est liée au thread principal et gère l'affichage du jeu ainsi que les contrôles (vérifier si un des joueurs a gagné, récupérer les commandes de l'adversaire sur le serveur, etc).

Dans cette même méthode est créé le thread secondaire de routine « routine_thread » qui gère le fonctionnement de l'IA et l'exécution des commandes. Les commandes sont exécutées directement après avoir été générées au sein même de l'IA, de sorte que chaque commande est générée selon les conséquences apportées par les précédentes.

6.1.2. Répartition sur différentes machines : connexion des joueurs

Pour jouer en réseau il nous faut 2 clients pour le serveur. Nous créons donc des services CRUD sur la donnée « user » via une API Web :

Requête GET /user/<id>	
Pas de données en entrée	
Cas joueur <id> existe	Statut OK Données sortie : type : « object », properties : { « name » : { type:string}, « creatures » : { type:string }, }, required : [« name », « creatures »]
Cas joueur <id> n'existe pas	Statut NOT_FOUND Pas de données de sortie

Dans tous les cas de figure /user/0 permet d'afficher le nombre de joueurs présents sur le serveur.

Requête PUT /user

Données en entrée :

```
type : « object »,
properties : {
        « name » : { type:string },
        « creatures » : { type:string },
},
required : [« name », « creatures »]
```

Cas il reste une place libre	Statut CREATED Données sortie : type : « object », properties : { « id » : { type:number,min :1,max :2 }, }, required : [« id »]
Cas il n'y a plus de place	Statut OUT_OF_RESOURCES Pas de données de sortie

Requête POST /user/<id>

Données en entrée :

```
type : « object »,
properties : {
        « name » : { type:string },
        « creatures » : { type:string },
},
required : [« name », « creatures »]
```

Cas joueur <id> existe	Statut OK Donnée de sortie type : « object », properties : { « name » : { type:string }, « creatures » : { type:string }, }, required : [« name », « creatures »]
Cas joueur <id> n'existe pas	Statut NOT_FOUND Pas de donnée de sortie

Requête DELETE /user/<id>

Pas de données en entrée

Cas joueur <id> existe	Statut NO_CONTENT Pas de données de sortie
Cas joueur <id> n'existe pas	Statut NOT_FOUND Pas de données de sortie

6.1.3. Répartition sur différentes machines : échange de commandes entre joueurs

En ce qui concerne la gestion des commandes, chaque joueur envoie ses commandes au serveur au moment où elles sont générées au sein de son IA. Afin d'envoyer les commandes au serveur, on dérive la classe engine::Engine par la classe client::SuperEngine (cf diagramme module.dia plus bas). Le joueur exécute alors normalement ses propres commandes avec son propre SuperEngine.

Lors de chacun de ses tours, le joueur va d'abord chercher sur le serveur les commandes exécutées dernièrement par son adversaire. Pour cela il a besoin de connaître via le serveur le nombre total de commandes qui ont été ajoutées au serveur, et le nombre total de commandes que son propre moteur a exécutées. Il les ajoute à sa liste de commande et les exécute. Ensuite son propre tour se déroule, et ainsi de suite.

La requête PUT sera exclusivement utilisée pour le dépôt des commandes sur le serveur :

Requête PUT /game

Données en entrée :

```
type : « object »,
properties : {
        «type» : { type:string },
        «player» : { type:string },
        // POUR LES COMMANDES DE DEPLACEMENT
        «initPos[0]» : { type:string },
        «initPos[1]» : { type:string },
        «finalPos[0]» : { type:string },
        «finalPos[1]» : { type:string },
        // POUR LES COMMANDES DE PLACE
        «finalPos[0]» : { type:string },
        «finalPos[1]» : { type:string },
        «creaType» : { type:string },
},
required : [«type», «player», «initPos[0]», «initPos[1]», «finalPos[0]», «finalPos[1]», «creaType»]
```

Sortie — Statut OK

Données sortie :

```
type : « object »,
properties : {
        « game.idmax » : { type:number,min :0,max :infini },
        },
required : [ « game.idmax » ]
```

La requête POST est utilisée pour indiquer au serveur quel joueur s'apprête à jouer.

Requête POST /game/2

Données en entrée :

```
type : « object »,
properties : {
        «player» : { type:number,min:1,max:2 },
},
required : [«player»]
```

<id> = 2	Statut OK Pas de données de sortie
Cas où <id>!= 2	Statut NOT_FOUND Pas de donnée de sortie

La requête GET permet de récupérer plusieurs choses :

<id>	**données**
0	Numéro de la partie tiré au sort par le serveur
1	Numéro du joueur qui commence tiré au sort par le serveur
2	Numéro du joueur dont c'est au tour de jouer
3	Nombre de commandes enregistrées sur le serveur
> 3	On récupère la commande placée à l'indice (id – 4) dans la liste des commandes du serveur

Requête GET /game/<id>

Pas de données en entrée

Cas <id> existe	Statut OK Pas de données de sortie
Cas <id> n'existe pas	Statut NOT_FOUND Pas de données de sortie

6.2. Sauvegarde d'un état du jeu

Pour sauvegarder l'état du jeu, on utilise le langage Json. On utilise ainsi un tableau de tableaux, dont le premier enregistre la position de chaque créature présente sur la carte - l'état initial étant généré aléatoirement - et les autres tableaux contiennent l'ensemble des commandes générées à chaque tour. Ces données sont ensuite stockées dans le fichier texte replay.txt.

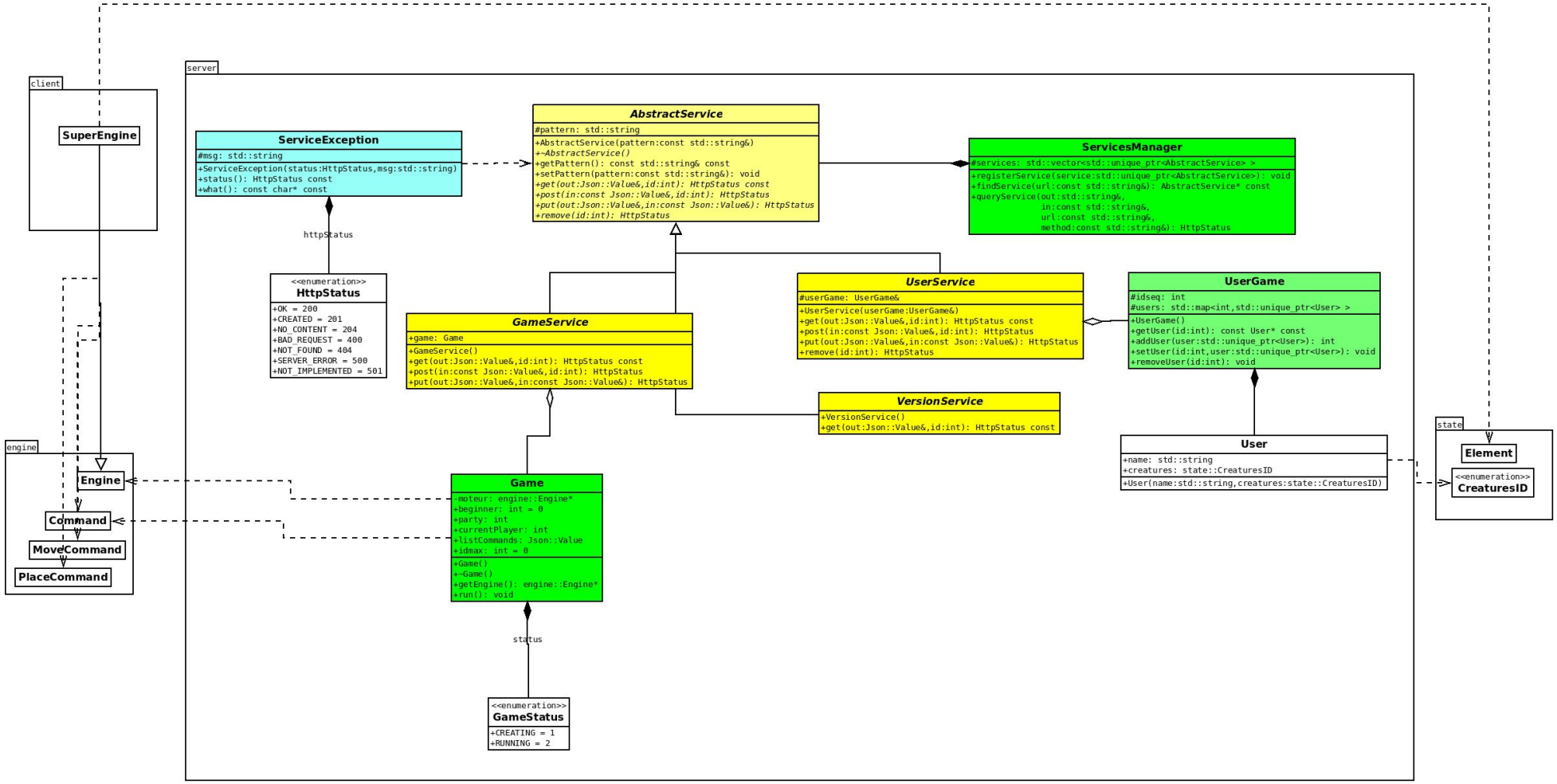

FIGURE 10 – Diagramme des classes de l’API Web

6.3. Conception logicielle

Voici l'utilité des classes définies sur le diagramme en figure 10 :

Game : La classe Game contient les éléments principaux d'une partie :

- beginner : numéro du joueur qui commence, tiré au hasard sur le serveur
- party : numéro de lapartie, tiré au hasard sur le serveur (entre 0 et 99)
- currentPlayer : numéro du joueur qui est en train de jouer
- listCommands : contient les données de toutes les commandes (sous format Json) jouées durant la partie
- idmax : nombre de commandes ajoutées au serveur
- le moteur de jeu présent dans la classe n'a finalement pas été utilisé

Les éléments du jeu sont régulièrement récupérés par les joueurs afin de modifier leur état du jeu.

Services : Les services REST sont implantés via les classes filles de AbstractService, et gérés par la classe ServicesManager :

- VersionService : renvoie la version actuelle de l'API
- UserService : permet d'ajouter, modifier, consulter et supprimer des joueurs. Permet en outre d'accéder au nombre de joueurs présents sur le serveur.
- GameService : fournit un service de consultation de l'état du jeu

Classe SuperEngine : classe fille de engine::Engine qui surcharge la méthode addCommand afin d'envoyer les données des commandes sur le serveur lorsqu'elles sont ajoutées à la liste de commandes du moteur du joueur.