Projet Logiciel Transversal

Gaétan RAYNAUD – Mathis OUDIN

Table des matières

1 Objectif

1.1 Présentation générale

Ce projet à pour but de la réalisation d'un jeu de type Battle for Wesnoth, avec des règles simplifiées.

1.2 Règles du jeu

Le jeu voit s'affronter deux équipes. Le jeu se déroule sur une carte découpée en cases, les cases sont réparties selon plusieurs types, ayant chacun leurs caractéristiques propres. Le jeu est découpé en tour, les joueurs jouent chacun leur tour.

Chaque équipe dispose d'une unité principale, le général, qui est l'unité principale de l'équipe. Si le général meurt, l'équipe perd la partie.

Chaque équipe commence avec une quantité d'or et peut en gagner à chaque début de tour en revendiquant une case de type maison, en plaçant une unité dessus. Si le général est situé sur une case de type château, il peut recruter d'autres unités de type épéiste, sur les cases de type caserne adjacentes.

A chaque tour, toutes les unités disposent d'une distance maximale de déplacement qui leur sont propre. Si une unité est sur une case adjacente à une unité ennemie, elle peut l'attaquer. Si une unité n'a plus de vie, elle meurt et disparaît du plateau.

1.3 Ressources

Le projet utilise plusieurs ressources pour l'affichage. Chaque texture est découpée en image de 72x72pixels.

Il y a huit textures pour les différents types de cases :

Figure 1: Texture des cases

Il y a six types d’unités, avec plusieurs animation pour les différentes actions, toutes les textures sont regroupées dans un même fichier, mais que l’on a découpé pour le mettre dans le rapport :

Figure 2: Texture du général humain

Figure 3: Texture de l'épéiste humain

Figure 4: Texture de l’archer humain

Figure 5: Texture du général orc

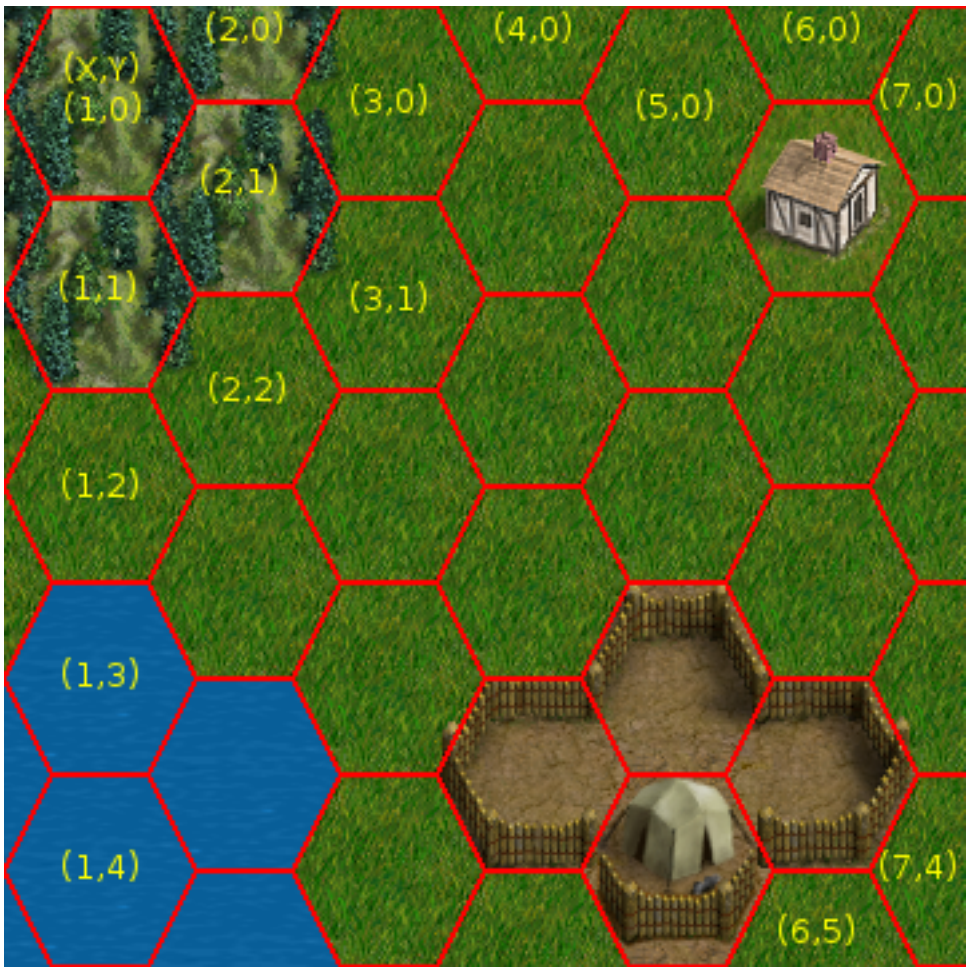
Figure 6: Texture de l'épéiste orc

Figure 7: Texture de l'archer orc

2 Description et conception des états

2.1 Description des états

Le jeu est formé d'une grille divisée en cases hexagonales avec différentes textures pour chaque case de la grille.



Sur cette grille des personnages mobiles se déplacent sur les cases à chaque tour. Les coordonnées (x, y) de chaque personnage correspondent à sa position dans la grille.

2.1.1 Éléments statiques : Terrain

La grille est formée de cases hexagonales de taille fixe. La grille est construite à partir de différentes cases possédant chacune une texture. La grille est générée au démarrage du niveau. Chaque case possède des coordonnées (x, y) de sa position dans la grille notée comme l'illustration ci-dessus, il ne peut y avoir qu'un seul terrain sur une position précise.
Chaque case possède un coup de déplacement pour les unités mobiles, spécifique à chaque type de case.
Les différents types de cases sont :

- **Les cases « Default » :** Les cases « Default » sont des éléments franchissables par les personnages et qui n'ont pas de propriétés particulières. Le choix du type change seulement le coût de déplacement du personnage ainsi que l'apparence de la case.
  Il existe plusieurs types de cases « Default » :
  - Les cases « Grass » classiques.
  - Les cases « Forest » avec une apparence de forêt.
  - Les cases « Water » qui représentent l'eau.
- **Les cases « House » :** Les cases « House » sont des éléments franchissables par les personnages. Lorsqu'un personnage termine son tour sur une case « House », celle-ci est revendiquée par l'équipe de l'unité. Chaque case revendiquée rapporte de l'or à l'équipe qui la possède au début de son tour.
- **Les cases « Castle » :** Les cases « Castle » sont des éléments franchissables par les personnages. Si un personnage « Leader » se situe sur une case « Castle », il peut alors recruter d'autres personnages pour son équipe. Soit des « Swordman » soit des « Bowman », en contrepartie d'un prix en or.
- **Les cases « Wall » :** Les cases « Wall » sont des éléments franchissables par les personnages. Les cases « Wall » sont associées à une case « Castle ». Lorsqu'une case « Wall » est collée à une case « Castle », le personnage « Leader » qui se troue sur la case « Castle », peut recruter un personnage pour son équipe sur les cases « Wall » vides. Le « Leader » ne peut recruter de personnage que sur les cases « Wall » ce qui permet de limiter le nombre de personnages recrutés à chaque tour.

2.1.2 Éléments mobiles : Unit

Les éléments mobiles possèdent des coordonnées qui correspondent à une position sur la grille, il ne peut y avoir qu'un seul élément mobile sur une position précise. Chaque unité possède également des points de vies, une vitesse correspondant au nombre de cases qu'il reste à l'unité pour se déplacer pendant son tour, un identifiant d'équipe, et une liste d'armes.
Les armes disponibles sont une épée et un arc.
Les différents types d'unités sont :

- **Les unités « Leader » :** Les unités « Leader » sont des unités spéciales, qui sont plus résistantes et puissantes que les unités « Swordman » et « Bowman ». Elles possèdent deux armes, un arc et une épée.
- **Les unités « Swordman » :** Les unités « Swordman » sont des unités basiques, qui possèdent uniquement une épée.
- **Les unités « Bowman » :** Les unités « Bowman » sont des unités plus avancées, qui possèdent un arc et une épée.

2.1.3 Éléments complémentaires : Weapon et Team

En plus des éléments statiques et mobiles, il existe encore deux types d'éléments.

- **Les « Team » :** Les éléments « Team » représentent les équipes du jeu, elles possède plusieurs attributs :
  - « gold » qui représente l'or que possède une équipe.
  - « nbHouse » qui indique le nombre de maisons revendiquées par l'équipe et permettra de savoir combien d'or l'équipe gagnera au début de son tour.
  - « race » qui définit la race à laquelle appartient l'équipe, humaine, ou orc, qui a pour but de différencier les deux équipes visuellement, avec des apparences différentes selon la race.
- **Les « Weapon » :** Les éléments « Weapon » sont les différents types d'armes disponibles, il en existe deux, l'arc et l'épée. Elles possèdent un unique attribut, « damage » qui représente les dégâts infligés par l'arme. Et une méthode « isCloseCombat » qui permet de savoir s'il s'agit d'une arme au corps-à-corps ou à distance.

2.1.4 Éléments généraux :

À tous les éléments déjà présents s'ajoute un élément global :

- **Le « Turn » :** L'élément « turn » est un compteur qui représente le numéro du tour de jeu en cours.

2.2 Conception logiciel

Pour ce jeu nous avons utilisé une structure présentée par la figure 8. Ce diagramme des classes, se découpe en plusieurs parties.

- **Les classes de type d'objets « Unit », « Weapon », « Terrain » et « Team » :** Ces quatre classes sont les classes qui définissent les différents types d'objets présents dans le jeu. Il y a quatre types d'objets, les « Unit » qui représentent les personnages gérés par le joueur. Les « Weapon » sont les armes équipées sur les différents personnages, chaque personnage possède sa ou ses propres armes. Les objets du type « Terrains » sont les objets représentant les éléments statiques du décor. Enfin les « Team » représentent les équipes, et contiennent les informations relatives à celles-ci.
- **Les classes objets « Sword », « Swordman », « Castle »… :** Les classes d'objets sont des classes dérivées des types d'objets. Ce sont des types avec des éléments spécifiques, comme le « Swordman » qui est une « Unit » avec une épée du type « Weapon », qui est différent d'un arc lui aussi du type « Weapon ». On les créé grâce à un système d'héritage, qui permet de regrouper dans « Weapon » toutes les caractéristiques communes à toutes les armes. Pour ensuite faire les spécificités dans « Sword » pour la différencier de « Bow ».
- **Les classes de conteneurs « Board » et « State » :** La troisième partie contient la classe « Board » qui est une classe qui contient tous les éléments du jeu. Elle est composée, de trois listes, une pour la liste des équipes du jeu, une pour la liste des unités, et une de tous les terrains disposés sur le plateau. Ce « Board » est un tableau à doubles entrées, de taille imposée, et chaque case du tableau correspond à une case du plateau de jeu. Une case contient un terrain, et peut contenir un personnage. La classe « State » est une classe représentant un état du jeu, composé d'un plateau.

2.3 Conception logiciel : extension pour le moteur de jeu

- **Les classes d'observateurs :** Les classes d'observateurs permettent de générer des événements à partir des modifications des éléments d'un état. Les observateurs récupèrent ces événements et peuvent ainsi s'adapter aux modifications des états. Ces classes suivent la paterne « Observer ».

2.4 Chargement de la carte

Pour la génération d'un niveau du jeu, c'est à dire les différents types de cases, on a choisi de passer par un fichier. Cela permet de facilement modifier le niveau et d'en créer de nouveaux rapidement.

Pour cela, on a ajouté une méthode « LoadTerrainsFromFile » à notre class « Board », à la quelle on donne en paramètre le chemin du fichier, et va remplir sa liste de terrains en fonction du fichier et avec les bonnes coordonnées.

Pour remplir ce fichier, il faut savoir plusieurs choses. Le première est la taille en nombre de tuile du niveau, que l'on retrouve dans le « board ». On remplit le fichier avec autant de caractères que l'on veut mais seul le nombre de tuile correspondant à la taille du niveau seront chargés. Le fichier se complète avec des caractères, chaque caractère correspondant à un type de terrain.
Les types de terrains sont :

- '0' : Herbe
- '1' : Forêt
- '2 : Eau
- '3' : Maison
- '4' : Château
- '5' : Mur gauche
- '6' : Mur haut
- '7' : Mur droit

2.5 Ressources

```
00000000111111222100000
00567000011122211131111
00040000000122011111111
00000000222220000000000
10100000232200000000000
11111111222010000060000
01031111221111100547000
00000112221111100000000
```

Figure 8: Exemple de fichier pour la génération d'un niveau

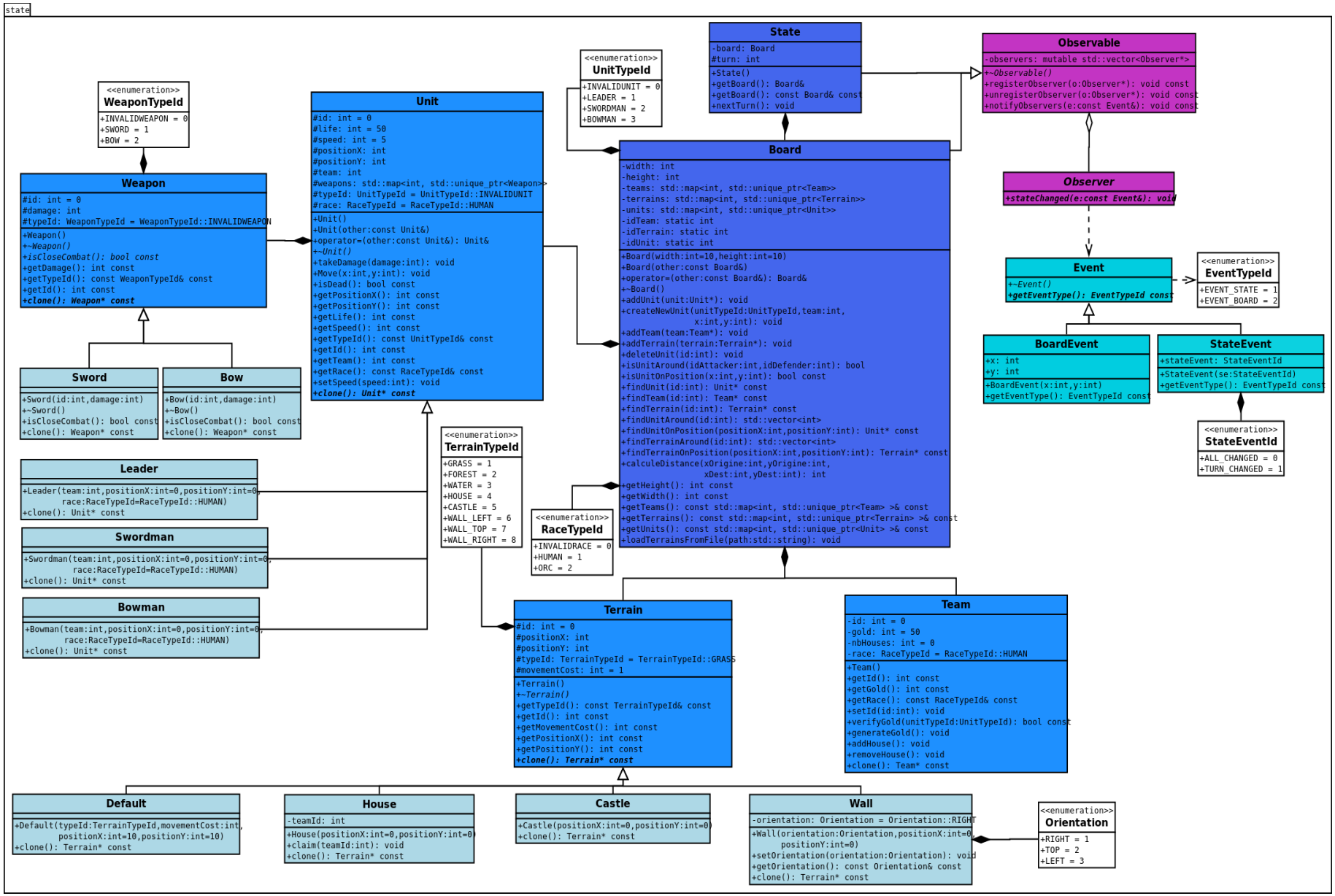


Figure 9: Diagramme des classes de l'état

3 Rendu : Stratégie et Conception

3.1 Stratégie de rendu d'un état

Pour notre projet nous avons opté pour un rendu d'état, le rendu ce fait à partir d'un état fixe calculé à l'avance. À chaque changement d'état le rendu est mis à jour.
Pour le rendu, nous découpons le jeu en trois plans (layers) :

- Un plan pour le le terrain avec les différentes cases (forêt, eau, château …),
- un plan pour les unités des deux équipes,
- un plan pour les menus.

Chaque plan est composé d'une unique texture. On récupère la texture de l'élément à afficher dans la texture du plan grâce à une tuile contenant sa taille et sa position dans l'image du plan.
Concernant la synchronisation on a une horloge pour le rendu qui est synchronisé à 30Hz. Pour l'instant, on met à jour le rendu en fonction de l'état à chaque coup d'horloge du rendu. Le but étant, lorsque l'on aura les événement de changement d'état, de ne mettre à jour le rendu qu'a chaque changement d'état. Le plan du terrain lui, n'est pas mit à jour. On le calcul au lancement du jeu, et ensuite il est toujours affiché de la même manière, car le terrain ne change pas au cours de la partie.

3.2 Conception logiciel

3.2.1 Génération des plans

La génération du rendu est découpée en plusieurs partie. La première, en jaune sur le diagramme en Figure 10, est la partie qui gère les plans. La classe « Layer » est la classe mère qui regroupe les informations générale d'un plan. De cette classe dérivent trois autres classes, une pour chaque plan du rendu. La surface associée à chaque classe, correspond au plan qui sera affiché. La méthode « initSurface » présente dans toutes les classes « Layer » est la méthode qui permet de créer la surface qui sera affichée en fonction soit de l'état, soit du « board ».

La classe Surface, contient plusieurs méthodes, qui permettent de charger le fichier de texture du plan concerné et ensuite de faire le lien entre les coordonnées entre les textures dans le fichier et les positions des différents éléments à l'écran. Pour cela on appelle la méthode « setSpriteLocation » avec la position d'origine (le coin supérieur gauche) ainsi que la taille de la texture pour la surface la place à l'écran.

Les classes « TileSet » sont les classes qui contiennent, les coordonnées des différentes textures dans les image des plan, par exemple dans la map terrains de « TerrainTileSet », il y a les coordonnées de toutes les textures des différents types de terrains.

Pour justement que la surface puisse savoir ou se trouve la texture à afficher dans le fichier d'image du plan, en ne connaissant que le type d'élément à afficher la surface fait appel aux « TileSet », elle envoie l'élément à afficher à la méthode « getTile ». Qui va lui renvoyer une « Tile » qui est un objet qui contient la position dans l'image de l'élément ainsi que sa taille, ce qui permettra à la surface de « copier » depuis le fichier à l'écran le partie de la texture correspondant à l'élément à afficher.

3.3 Conception logiciel : extension pour les animations

3.4 Ressources

3.5 Exemple de rendu

Figure 10: Exemple de rendu

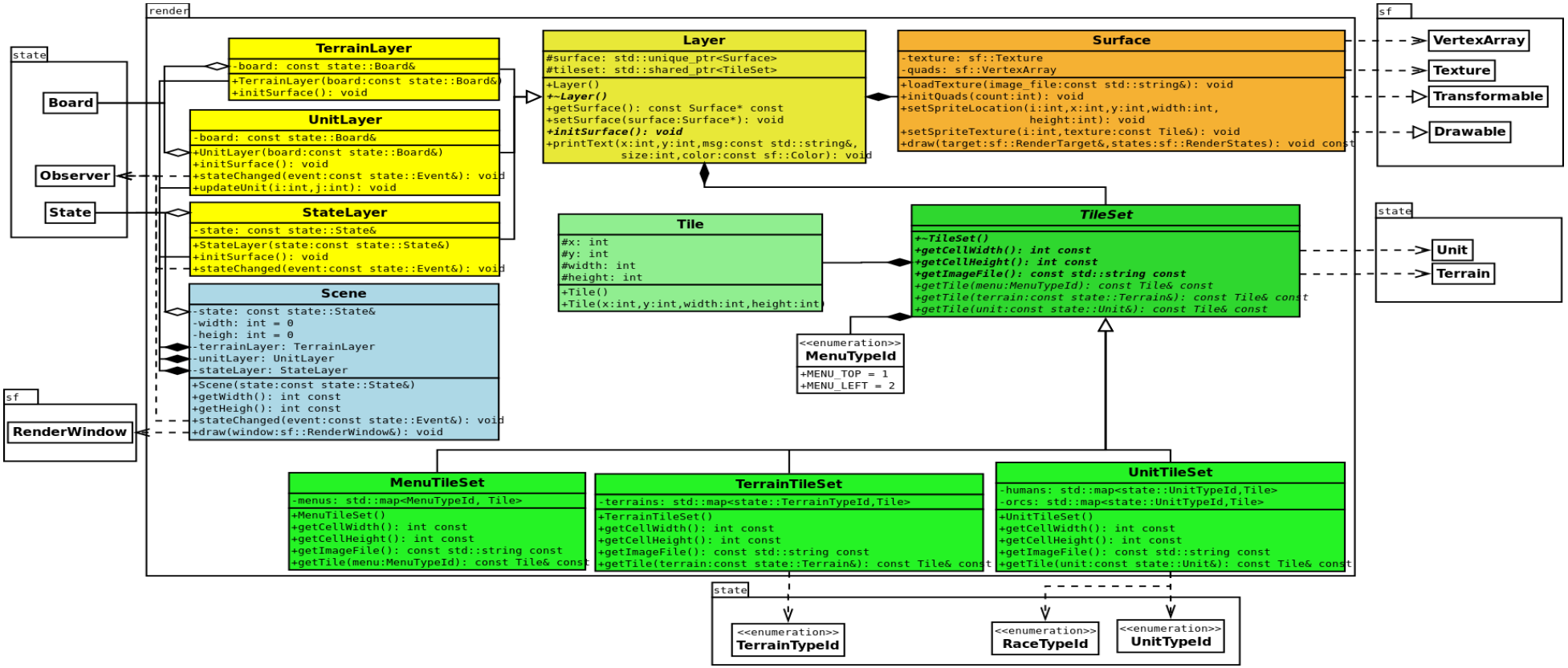


Figure 11: Diagramme des classes pour le rendu

4 Règles de changement d'états et moteur de jeu

4.1 Horloge globale

Les changements d'état suivent une horloge globale : de manière régulière, on passe directement d'un état à un autre.

4.2 Changements extérieurs

Les changements extérieurs sont provoqués par des commandes extérieurs, comme la pression sur une touche ou un ordre provenant du réseau, toutes les commandes extérieurs sont :

- **« Charger un niveau » :** On fabrique un état initial à partir d'un fichier
- **« Déplacer un personnage » :** Elle prend en paramètre l'id du personnage à déplacer ainsi que les positions de la destination, et déplace le personnage vers la position indiquée
- **« Attaquer avec un personnage » :** Elle prend en paramètre l'id du personnage qui attaque et de celui attaqué, ainsi que l'arme avec laquelle le personnage attaque, et inflige les dégâts de l'arme au personnage attaqué
- **« Recruter un personnage » :** Elle prend en paramètre la position du personnage qui recrute ainsi que le type de personnage à recruté. Si le recruteur est un « Leader » et est sur une case de type « Castle », il instancie un nouveau personnage pour son équipe sur une case adjacente de type « Wall ».

4.3 Conception logiciel

Le moteur du jeu, est basé sur le paterne « Command », et permet d'exécuter une série d'action de manière différée.

- **Les classes « Command » :** Le rôle de ces classes est de représenter une commande, chaque classe fille représente une commande différente, que l'on repère grâce à leur « CommandTypeId » :
  - **LOAD :** Charger un niveau
  - **MOVE :** Déplacer un personnage
  - **ATTACK :** Attaquer avec un personnage
  - **SPAWN :** Recruter un personnage

- **Engine :** C'est la classe qui va contenir toutes les commandes dans sa « map », la « map » permet de donner un indice de priorité à chaque commande. A chaque coup de l'horloge globale, on appel la méthode « update » du moteur, qui fait elle même appelle à la méthode « execute » de chaque commande enregistrée, puis vide la liste.

4.4 Conception logiciel : extension pour l'IA

4.5 Conception logiciel : extension pour la parallélisation

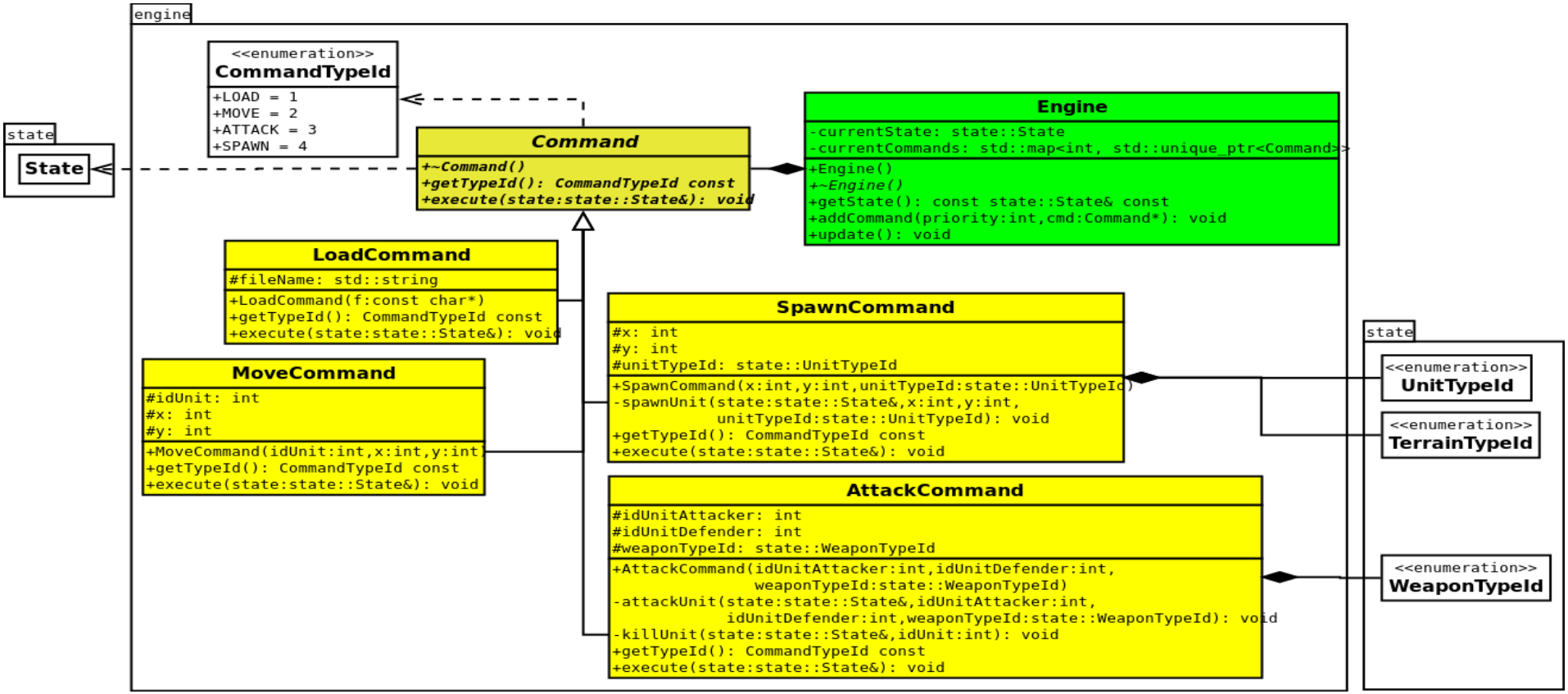


Figure 12: Diagramme des classes du moteur de jeu

5 Intelligence Artificielle

Cette section est dédiée aux stratégies et outils développés pour créer un joueur artificiel. Ce robot doit utiliser les mêmes commandes qu'un joueur humain, ie utiliser les mêmes actions/ordres que ceux produit par le clavier ou la souris. Le robot ne doit pas avoir accès à plus information qu'un joueur humain. Comme pour les autres sections, commencez par présenter la stratégie, puis la conception logicielle.

5.1 Stratégies

5.1.1 Intelligence minimale

5.1.2 Intelligence basée sur des heuristiques

5.1.3 Intelligence basée sur les arbres de recherche

5.2 Conception logiciel

5.3 Conception logiciel : extension pour l'IA composée

5.4 Conception logiciel : extension pour IA avancée

5.5 Conception logiciel : extension pour la parallélisation

6 Modularisation

Cette section se concentre sur la répartition des différents modules du jeu dans différents processus. Deux niveaux doivent être considérés. Le premier est la répartition des modules sur différents threads. Notons bien que ce qui est attendu est un parallélisation maximale des traitements: il faut bien démontrer que l'intersection des processus communs ou bloquant est minimale. Le deuxième niveau est la répartition des modules sur différentes machines, via une interface réseau. Dans tous les cas, motivez vos choix, et indiquez également les latences qui en résulte.

6.1 Organisation des modules

6.1.1 Répartition sur différents threads

6.1.2 Répartition sur différentes machines

6.2 Conception logiciel

6.3 Conception logiciel : extension réseau

6.4 Conception logiciel : client Android

Illustration 1: Diagramme de classes pour la modularisation