



Projet Logiciel Transversal

Luigi CAPO-CHICHI – Yassert BOINALI

Table des matières

1	Objectif.....	3
1.1	Présentation générale.....	3
1.2	Règles du jeu.....	3
1.3	Ressources.....	3
1.3.1	Textures pour le déco.....	3
1.3.2	Textures pour les unités militaires.....	4
1.3.3	Textures pour les batiments.....	4
1.3.4	Textures pour les ressources.....	5
1.3.5	Textures pour les technologies.....	5
2	Description et conception des états.....	6
2.1	Description des états.....	6
2.1.1	Etat des éléments fixes.....	6
2.1.2	Etat des éléments mobiles.....	6
2.1.3	Etat général.....	6
2.2	Conception logiciel.....	7
3	Rendu : Stratégie et Conception.....	9
3.1	Stratégie de rendu d'un état.....	9
3.2	Conception logiciel.....	9
3.3	Conception logiciel : extension pour les animations.....	9
3.4	Ressources.....	9
3.5	Exemple de rendu.....	9

1 Objectif

1.1 Présentation générale

Notre projet consiste à réaliser une version simplifiée du jeu de stratégie au tour par tour « Civilization ».

1.2 Règles du jeu

Le joueur doit développer son empire en compétition avec une ou plusieurs autres civilisations dirigées par l'ordinateur (IA). Le but du jeu est d'avoir la plus importante civilisation quand le jeu s'arrête. A son tour, le joueur peut déplacer ses pièces, attaquer, échanger, découvrir de nouvelles technologies et construire de nouvelles unités militaires, colons et colonies.

Le jeu se déroule sur trois époques. La plus ancienne est l'antiquité, suivi de l'ère industrielle et finalement de l'ère moderne. Chaque époque a ses propres forces militaires, améliorations de villes et technologies, et chacune est supérieure à celle de l'ère précédente.

Le jeu débute à l'ère antique. Une époque se finit quand :

- un joueur achète la 3e technologie de l'ère actuelle, ou
- un joueur achète la dernière technologie restante de l'ère actuelle.

En combinant habilement le développement économique, la force militaire, la diplomatie et les échanges profitables, le joueur peut créer une plus grande civilisation et gagner la partie.

Fin du jeu

Le jeu se termine à la fin du tour où un joueur possède 3 technologies de l'ère moderne. Quand tous les joueurs ont fini leurs achats, on compte les points de victoire. Le joueur avec le plus de points de victoire a gagné.

1.3 Ressources

L'affichage repose sur plusieurs textures qu'on va présenter ci-dessous.

1.3.1 Textures pour le décor



- Fond écran du jeu



- Terrain

Désert	Montagne	Plaine	Océan

1.3.2 Textures pour les unités militaires

	Armées			Véhicules	
	Infanterie	Cavalerie	Artillerie	Flottes	Aviation
Epoque Antique	Epéiste	Cavalier	Catapulte	Galère	-
					
Epoque Industrielle	Mousquet	Dragon	Canon	Frégate	-
					
Epoque Moderne	Mitrailleur	Tank	Obusier	Cuirassé	Chasseur
					

1.3.3 Textures pour les bâtiments

- Bâtiments antiques



- Bâtiments industrielles



- Métropole (4 fois colonie)



1.3.4 Textures pour les ressources

Vin	Chevaux	Fer	Epices	Charbon	Métaux précieux
					

1.3.5 Textures pour les technologies



2 Description et conception des états

2.1 Description des états

Un état du jeu est formée par un ensemble d'éléments fixes (plateau, ressources, colonies, murs) et un ensemble d'éléments mobiles (colons, militaires, catapulte). Tous les éléments possèdent les propriétés suivantes :

- Coordonnées (x,y) dans la grille
- Identifiant du type d'élément : ce nombre indique la nature de l'élément (c'est à dire de la classe).

2.1.1 Etat des éléments fixes

Le plateau est formé par une grille d'éléments nommé « cases ». La taille de cette grille est fixe. Les types de cases sont :

Cases « Mur » : les cases « mur » sont des éléments infranchissables pour les éléments mobiles.

Cases « Plateau » : les cases « Plateau » vont servir à définir la texture du terrain (désert, plaine, montagne, océan).

Cases « Ressource » : elles contiennent un type de ressource (cheval, charbon, fer, vin).

Cases « Colonie » : ces cases vont contenir les différents types de bâtiments.

2.1.2 Etat des éléments mobiles

Les éléments mobiles possèdent une direction (aucune, gauche, droite, haut ou bas) et un coût. Chaque élément mobile peut se déplacer d'un certain nombre de cases prédéfini (PM).

Element mobile « Colon » : cet élément est dirigé par le joueur, qui commande la propriété de direction. Les colons sont les seules pièces qui peuvent explorer les régions terrestres et bâtir des colonies.

Elements mobiles « Militaire » et « Catapulte » : ces éléments sont également commandés par la propriété de direction, qu'elle proviennent d'un humain ou d'une IA. Ces éléments disposent d'un point de vie prédéterminé qui évolue lors des combats.

2.2 Conception logiciel

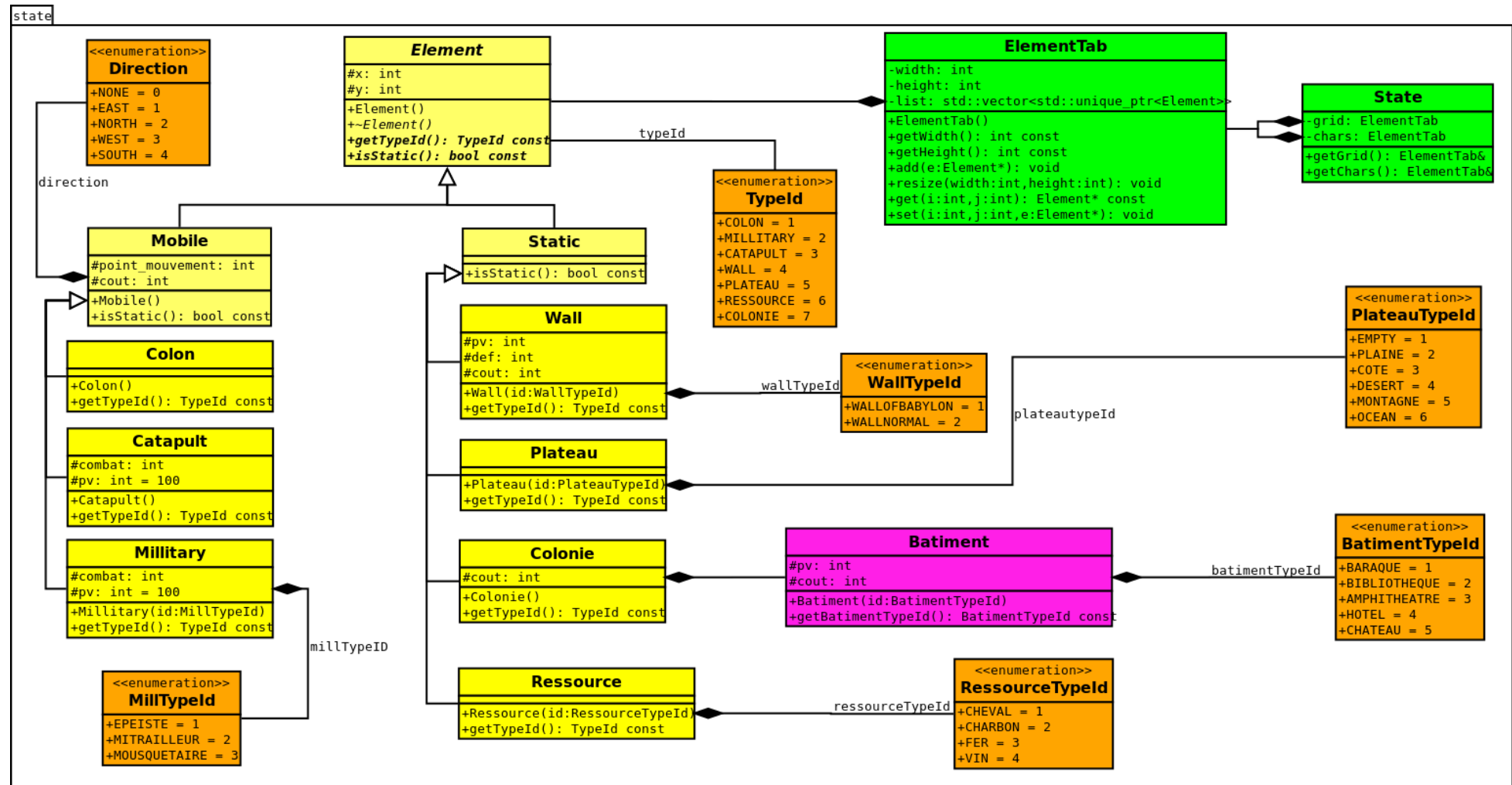
Le diagramme des classes pour les états est présenté sur la figure ci-dessous dont nous pouvons mettre en évidence les groupes de classes suivants :

Classes Element : on dispose de toute une hiérarchie de classes filles qui héritent de la classe « Element » (en jaune). Ces classes permettent de représenter les différents catégories et types d'élément.

Conteneurs d'élément : viennent ensuite les classes State et ElementTab qui permettent de contenir des ensembles d'éléments. ElementTab est un tableau en deux

dimension d'éléments, par exemple pour contenir la grille des éléments. Il peut également être considéré comme un tableau à une dimension dans le cas où la hauteur (height) est égale à 1 (utile pour la liste des personnages « chars »). Enfin, la classe State est le conteneur principal, à partir duquel on peut accéder à toutes les données de l'état.

Figure 1: Diagramme des classes d'état



3 Rendu : Stratégie et Conception

3.1 Stratégie de rendu d'un état

Afin de pouvoir réaliser un rendu imagé de notre jeu Civilization, il nous faut tout d'abord réaliser un rendu des différents éléments qui composent notre jeu. Pour cela, on adopte une stratégie assez bas niveau et relativement proche du fonctionnement des unités graphiques.

Plus précisément, nous découperons la scène à rendre en layers(plans). On aura ainsi un plan pour les éléments statiques (Wall, Colonie, Batiment...), un autre pour les éléments mobiles tels que les unités militaires et les colons puis enfin un plan pour le score et l'arbre des technologies. Chaque plan contiendra deux informations de bas niveau à savoir une unique texture contenant les tuiles et une unique matrice avec la position des éléments et les coordonnées dans la texture. Ainsi, seuls les éléments dont les tuiles sont associés à la texture du plan seront réalisés dans notre rendu.

Pour la réalisation de ces différentes informations, la première idée est d'observer l'état à rendre et de réagir, lorsqu'une modification se produit. Si on observe un changement permanent dans le rendu, on met à jour le morceau de la matrice du plan correspondant. Pour les changements non permanents comme les éléments mobiles, il faudra modifier la matrice du plan automatiquement à chaque rendu d'une nouvelle frame.

3.2 Conception logiciel

Le diagramme d'état pour le rendu général est présenté en figure 2.

Layer: Cette classe est le moteur du rendu qu'on veut réaliser. Le principal but des objets de cette classe est de former des éléments basiques qu'on pourra transmettre à notre carte graphique par l'intermédiaire d'une classe surface. Les classes filles de cette classe permettent de définir les différents plans de notre jeu. Ainsi la classe StateLayer permettra de réaliser le plan contenant les différentes informations, et la classe ElementTabLayer constituera par exemple un plan pour les différents personnages de notre jeu.

La méthode initSurface() de cette classe permettra de créer une surface, de changer sa texture et d'initialiser la liste des sprites.

Tuiles: La classe TileSet permet de définir les tuiles des différents éléments de notre jeu. Ces classes filles regroupent toutes les définitions des tuiles d'un même plan. La classe StateTileSet pour les informations au niveau du jeu, la classe GridTileSet pour les éléments statiques et la classe CharsTileSet pour les éléments mobiles.

Surface: Chaque surface contient une texture du plan et une liste de quadruplet de vecteurs.

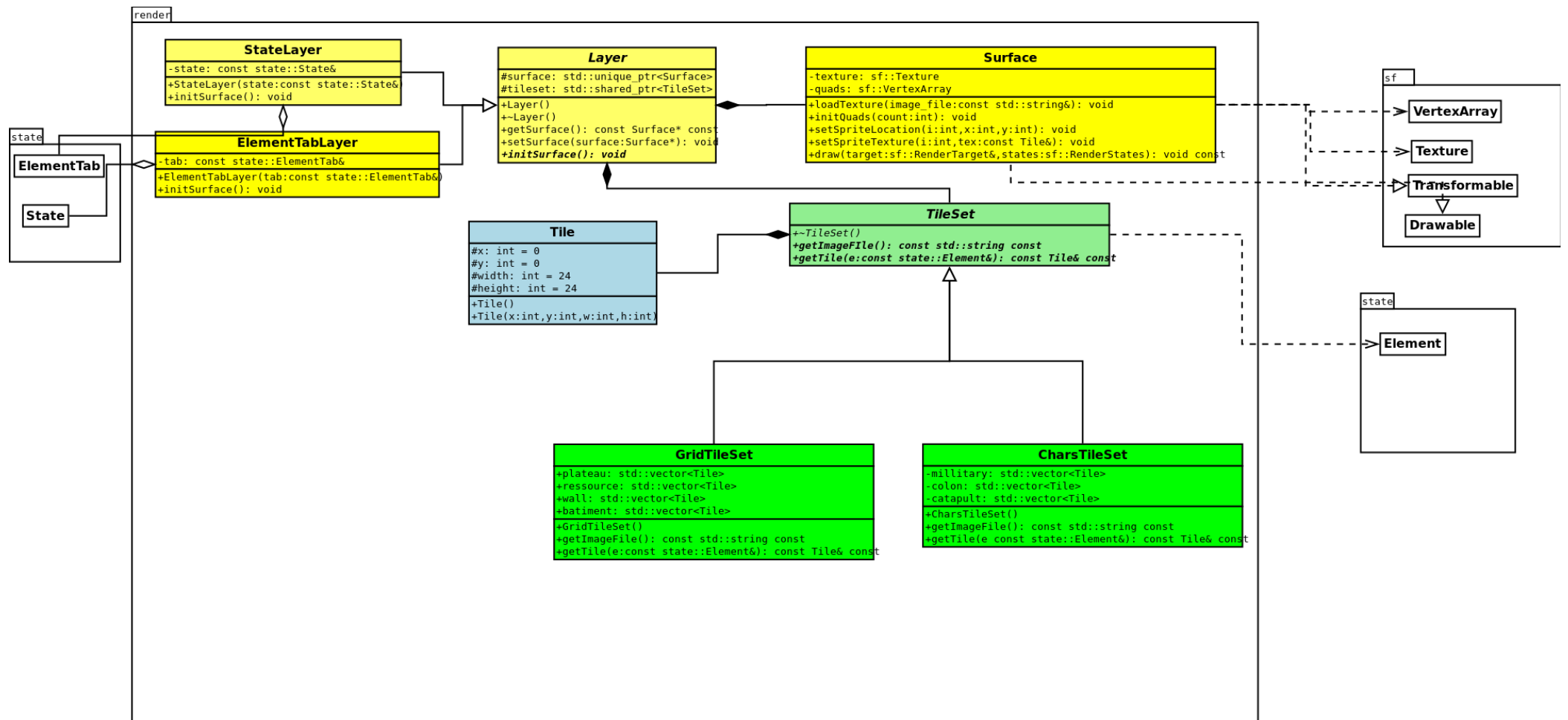


Figure 2: Diagramme de classes pour le rendu