Projet Logiciel Transversal

Luigi CAPO-CHICHI – Yassert BOINALI

Table des matières

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 déco

- 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
					

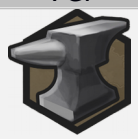
1.3.3 Textures pour les batiments

- Batiments antiques

- Batiments 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 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 (desert, 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 batiments.

2.1.2 Etat des éléments mobiles

Les éléments mobiles possède une direction (aucune, gauche, droite, haut ou bas), une vitesse et un coût. Une position à zéro signifie que l'élément est exactement sur la case ; pour les autres valeurs, cela signifie qu'il est entre deux cases (l'actuelle et celle définie par la direction de l'élément). Lorsque la position est égale à la vitesse, alors l'élément passe à la position suivante.
Chaque élément mobile peut se déplacer d'un certain nombre de cases prédifini (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 fixée qui évolue lors des combats.

2.1.3 Etat général

A l'ensemble des éléments statiques et mobiles, nous rajoutons les propriétés suivantes :

- Epoque : représente « l'heure » correspondant à l'état, c'est le nombre de « tic » de l'horloge globale depuis le début de la partie.
- Vitesse : le nombre d'époque par seconde, c'est à dire la vitesse à laquelle l'état du jeu est mis à jour.

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 :

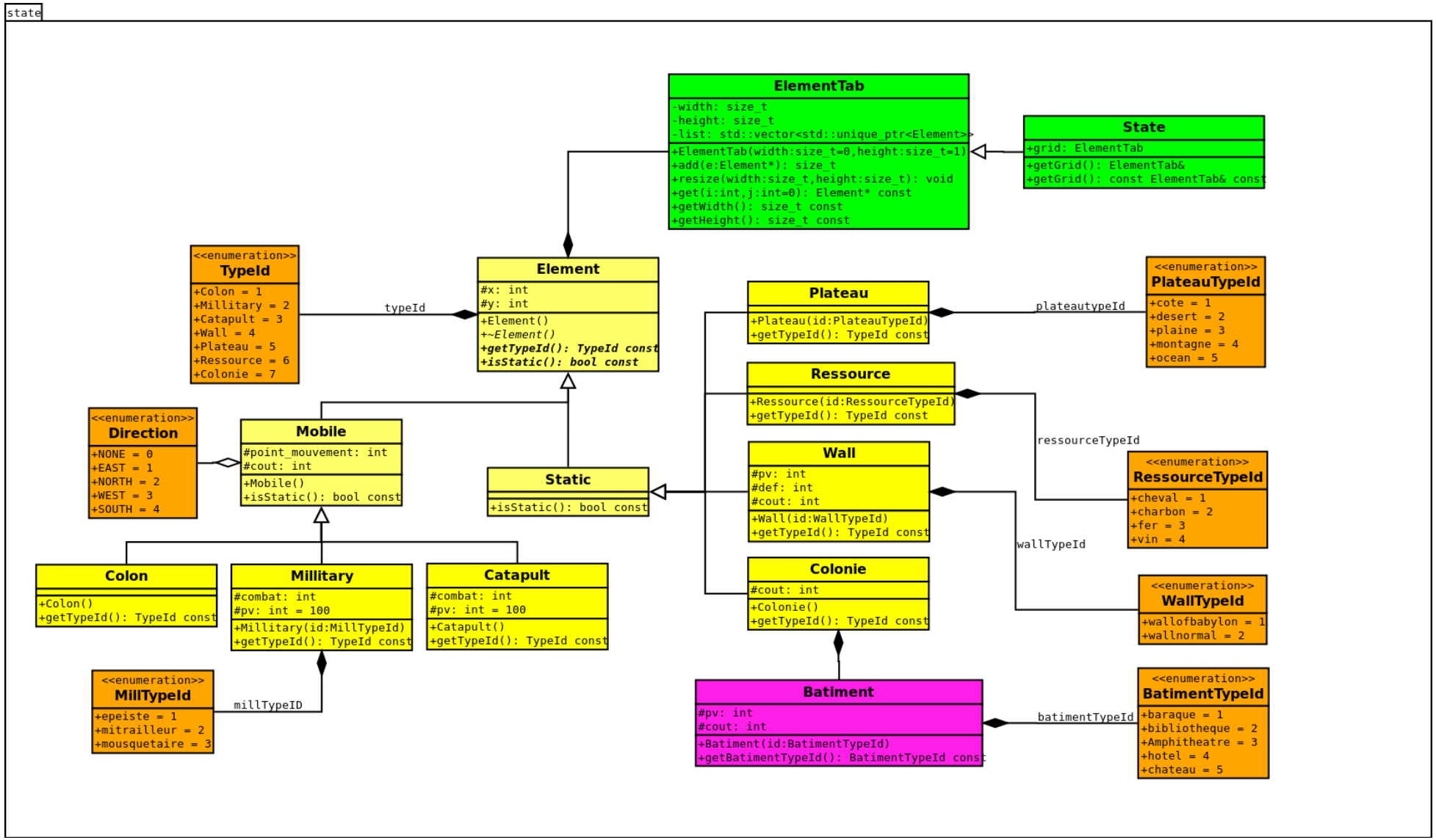
Classes Element : on dispose de toute une hiérarchie de classes filles qui héritent de la classe « Element ». Ces classes permettent de représenter les différents catégories et types d'élément.

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 l'arbre des technologies...). Enfin, la classe State est le conteneur principal, à partir duquel on peut accéder à toutes les données de l'état.

Fabrique d'éléments : Dans le but de pouvoir fabriquer facilement des instances d'Element, nous utilisons la classe ElementFactory. Cette classe, qui suit le patron de conception Abstract Factory, permet de créer n'importe quelle instance non abstraite à partir d'un caractère. L'idée est de l'utiliser, en autres, pour créer un niveau à partir d'un fichier texte. Par exemple, au caractère « - », on fait correspondre une instance de Wall avec la propriété « wallTypeId » égale à HORI. Dans le but de faciliter l'enregistrement des différentes instancation possible, nous avons créé le couple de classes AElementAlloc et ElementAlloc<E,ID>. La première est une classe abstraite dont le seul but est de renvoyer une nouvelle instance. La seconde sert à créer des implantations de la première pour une classe et une propriété d'identification particulière. Par exemple, pour créer le créateur de mur horizontal et l'enregister dans une factory, il suffit de faire :

```
factory.registerType( '− ' , newElementAlloc<Wall , WallTypeId > (HORI )) ;
```

Figure 1: Diagramme des classes d'état



3 Rendu : Stratégie et Conception

Présentez ici la stratégie générale que vous comptez suivre pour rendre un état. Cela doit tenir compte des problématiques de synchronisation entre les changements d'états et la vitesse d'affichage à l'écran. Puis, lorsque vous serez rendu à la partie client/serveur, expliquez comment vous aller gérer les problèmes liés à la latence. Après cette description, présentez la conception logicielle. Pour celle-ci, il est fortement recommandé de former une première partie indépendante de toute librairie graphique, puis de présenter d'autres parties qui l'implémente pour une librairie particulière. Enfin, toutes les classes de la première partie doivent avoir pour unique dépendance les classes d'état de la section précédente.

3.1 Stratégie de rendu d'un état

3.2 Conception logiciel

3.3 Conception logiciel : extension pour les animations

3.4 Ressources

3.5 Exemple de rendu

Illustration 2: Diagramme de classes pour le rendu