Projet Advance Wars

Ahmed BEN HAMOUDA – Benjamin FRANÇOIS

FIGURE 1 – Exemple du jeu Advance Wars

Table des matières

1 Objectif

1.1 Archétype

L'objectif de notre projet est de développer une version simplifiée du jeu Advance Wars. Afin de personnaliser notre projet, nous avons décidé de créer une variante des règles originales du jeu.

1.2 Règles du jeu

On dispose d'un terrain composé de deux bases avec plusieurs unités. Chaque joueur contrôle les unités de sa base. Contrairement au jeu original où le but est de capturer la base ennemie, l'objectif ici va être de capturer un drapeau situé dans la base ennemie pour la ramener dans sa propre base. Le vainqueur est celui qui aura ramené le drapeau ennemi dans sa propre base le premier.
Le jeu se déroule au tour par tour : lorsqu'un joueur attaque, il peut déplacer chacune de ses unités individuellement. Si une de ses unités rencontre une unité ennemie, le joueur a la possibilité de lancer une offensive. Enfin, pour éviter de perdre toutes ses unités au combat, chaque joueur possède une solde qui augmente à chaque tour. De cette manière, il peut acheter des unités supplémentaires. Il est à noter qu'il existe différents types d'unités, et que les unités plus puissantes coûtent plus cher que les unités de base.

1.3 Ressources

L'affichage du jeu repose principalement sur trois textures :

- une texture de 464 x 48 pixels correspondant aux tuiles du terrain, présentée dans la figure 2.
- une texture de 287 x 80 pixels correspondant aux tuiles des différentes unités, présentée dans la figure 3.
- une texture de 97 x 195 pixels correspondant aux tuiles des bâtiments, présentée dans la figure 4.

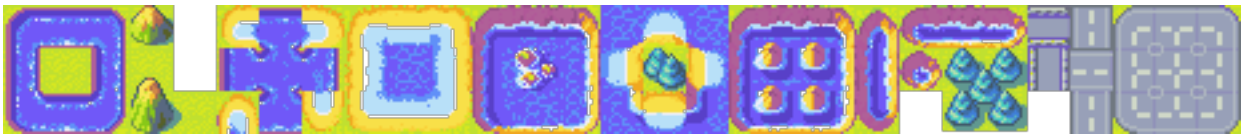

FIGURE 2 – Texture pour les tuiles du terrain

FIGURE 3 – Texture pour les tuiles des unités

FIGURE 4 – Texture pour les tuiles des bâtiments

2 Description et conception des états

2.1 Description des états

L'état du jeu est formée par trois couches d'éléments :
— La couche inférieure constituée du terrain.
— La couche intermédiaire constituée des bâtiments,
— La couche supérieure formée par un ensemble d'éléments mobiles (les unités).

2.1.1 Couche inférieure

La couche inférieure correspond à un tableau en 2D d'éléments de « type de terrain ». Un type de terrain correspond à un élément de décor présent sur chaque case de la grille de jeu. Il existe 5 types de terrain :
— Les routes. Celles-ci peuvent être pratiquées par toutes les unités, elles favorisent le déplacement des véhicules.
— Les plaines. Toutes les unités peuvent se déplacer sur celles-ci.
— Les forêts. Les véhicules peuvent difficilement se déplacer sur ces types de terrains.
— Les montagnes. Les véhicules lourds ne peuvent pas s'y déplacer.
— Les villes. Les unités sont affectés de la même manière que les forêts.

2.1.2 Couche Intermédiaire

La couche intermédiaire est constituée par des bâtiments, des éléments qui sont fixes sur le terrain. Ces bâtiments possèdent les propriétés suivantes :
— Position (x,y) dans la grille.
— Color (c) représentant la couleur des équipes qui s'affrontent.
— Id_b(id_b) représentant l'identité de l'unité pour l'affichage dans le rendu.
Les bâtiments sont composé de deux types d'éléments :
— **Le bâtiment principal.** Cet élément contient l'objet à récupérer par le joueur adverse (le drapeau).
— **Les bâtiments secondaires.** Ces éléments permettent la construction des unités sur la même position où le bâtiment se trouve à condition qu'aucune autres unité, allié ou ennemie, ne se trouve sur le bâtiment.

2.1.3 Couche Supérieure

La couche supérieure est constituée par les unités, les éléments mobiles sur le terrain. Ces unités possèdent les propriétés suivantes :
— Position (x,y) dans la grille.
— Color (c) représentant la couleur des équipes qui s'affrontent.
— Vie(v) représentant la vie de l'unité.
— Mvt (m) représentant les points de mouvement de l'unité.
— Puissance (p) représentant la puissance de l'unité.
— Prix (prix) représentant le prix d'achat de l'unité.
— Id(id) représentant l'identité de l'unité pour l'affichage dans le rendu.

2.1.4 État général

L'état général du jeu, en plus du terrain composé des trois couches décrites précédemment, est également défini par deux variables :
— Le tour. Cette variable sert à savoir combien de tours ont été joués depuis le début du jeu et à

déterminer à quel joueur c’est le tour de jouer.
— La fin. Cette variable sert à déterminer si la partie est terminée ou non.

2.2 Conception Logiciel

Le diagramme des classes pour les états est présenté en Figure 5, dont nous pouvons mettre en évidence les groupes de classes suivants :

Classe Unite. Cette classe, étant polymorphe, sert à définir les types d’unités différentes, représentées par les classe filles. Chaque classe dérivée de la classe **Unité** possède des valeurs prédéfinies pour ses différents attributs, correspondant aux caractéristiques du type d’unité. Pour placer les unités sur la grille de jeu, on les stocke dans un tableau à deux dimensions de type vector<vector<Unite*>> (on stocke les pointeurs pour profiter du polymorphisme de la classe). Les coordonnées des unités dans le tableau correspondent à leur position sur la grille. Cette position est définie par la classe **Position**.

Classe Batiment. Cette classe représente les bâtiments servant à fabriquer les unités. Cette classe est donc développé selon le design pattern **Factory**. Comme pour la classe **Unite**, on stocke les instances de **Batiment** dans un tableau 2D.

Classe TerrainTab. Cette classe correspond à un tableau en 2D des éléments de **TerrainTypeId**, une structure de données de type énumération qui sert à représenter les types de terrain.

Conteneurs d’éléments. On retrouve les classes **Terrain** et **Jeu**. La première classe contient les trois tableaux 2D correspondant aux trois couches d’éléments de la grille de jeu que nous avons décrite précédemment. Elle contient également des méthodes de type « getter » qui permettent de renvoyer les éléments présents sur la grille en fonction de leur position. Enfin, la classe **Jeu** contient une occurrence de la classe **Terrain**.

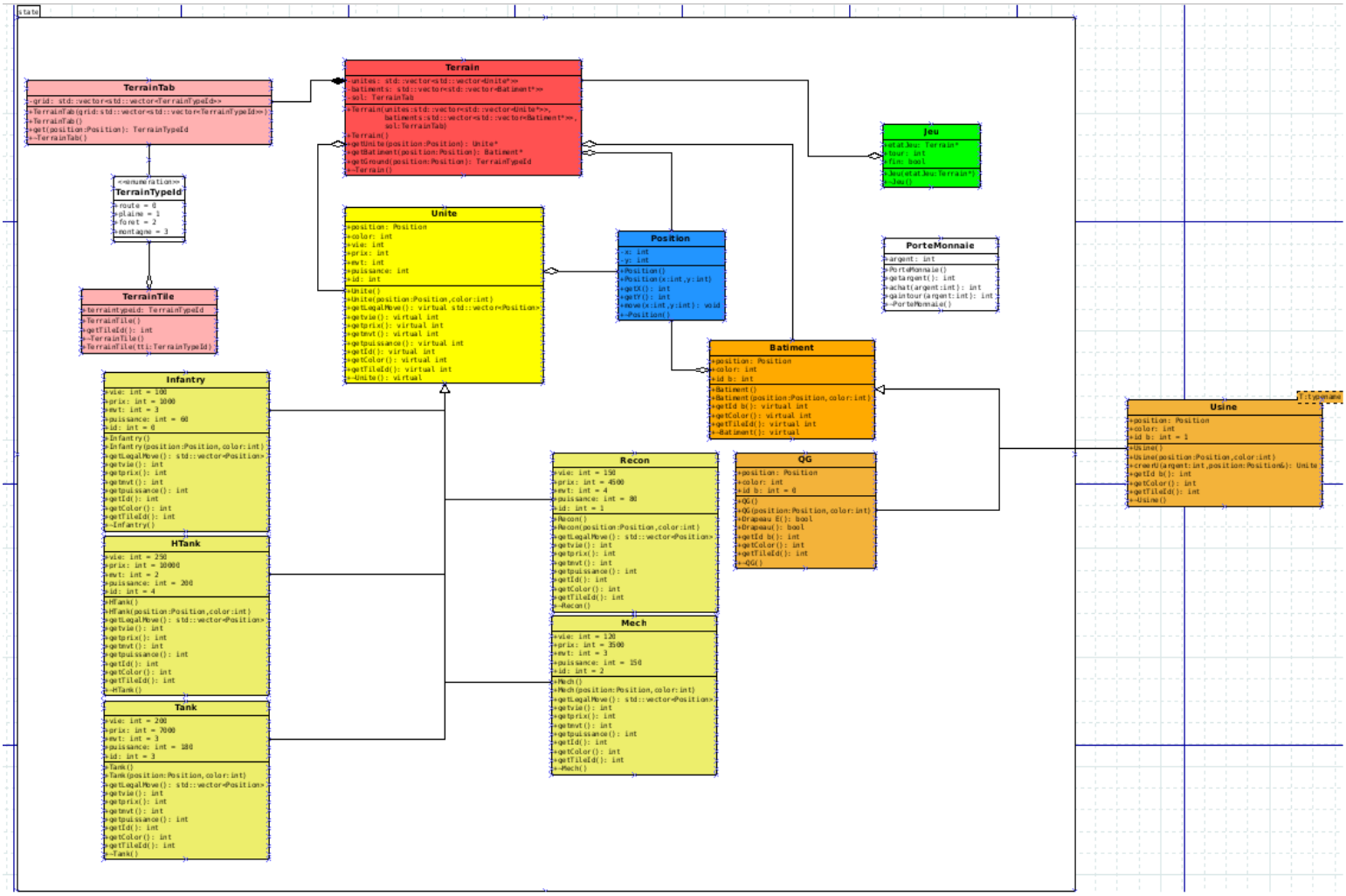

FIGURE 5 – Diagramme UML de l'implémentation de l'état du jeu

3 Rendu : Stratégie et Conception

3.1 Stratégie de rendu d'un état

Pour le rendu d'un état, nous avons opté pour une stratégie assez bas niveau, et relativement proche du fonctionnement des unités graphiques.
Plus précisément, nous découpons la scène à rendre en plans (ou « layers ») : un plan pour le décors (plaine, foret, etc.), un plan pour les bâtiments (QG, usines) et un plan pour les éléments mobiles (les unités). Chaque plan contiendra deux informations bas-niveau qui seront transmises à la carte graphique : une unique texture contenant les tuiles (ou « tiles »), 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.

3.2 Conception logiciel

Le diagramme des classes pour le rendu général est présenté en Figure 6.
Plan :
Le cœur du rendu réside dans la classe Layers. Le principal objectif d'une instance de Layer est de donner les informations basiques pour former les éléments bas-niveau à transmettre à la carte graphique. Ces informations sont données à une instance de Surface, et la définition des tuiles est contenu dans une instance de TileSet. Les classes filles de la classe Layer se spécialise pour l'un des plans à afficher. Pour chaque plan, on a une méthode nommée setSurface() qui fabrique une nouvelle surface, lui demande de charger la texture, puis initialise la liste des sprites. Par exemple, pour afficher le niveau, elle demande un nombre de quads/sprites égal aux nombre de cellules dans la grille avec initQuads(). Puis, pour chaque cellule du niveau, elle fixe leur position avec setSpriteLocation() et leur tuile avec setSpriteTexture().
Surface :
Chaque surface contient une texture du plan et une liste de paires de quadruplets de vecteurs 2D. Les éléments texCoords de chaque quadruplet contient les coordonnées des quatre coins de la tuile à sélectionner dans la texture. Les éléments position de chaque quadruplet contient les coordonnées des quatre coins du carré où doit être dessiné la tuile à l'écran.
Tuiles :
Le patern Factory TileSet regroupent toutes les définitions des tuiles utilisé pour chaque plan à part. Par exemple, il sait que les coordonnées de la tuile avec un QG vert qui est (0,93) et de taille (16,31). Pour obtenir une telle information, un client de ces classes utilise la méthode getTile(). Par exemple si on passe à cette méthode une instance de la classe QG, alors elle va renvoyer une instance de Tile qui correspond à cet élément.

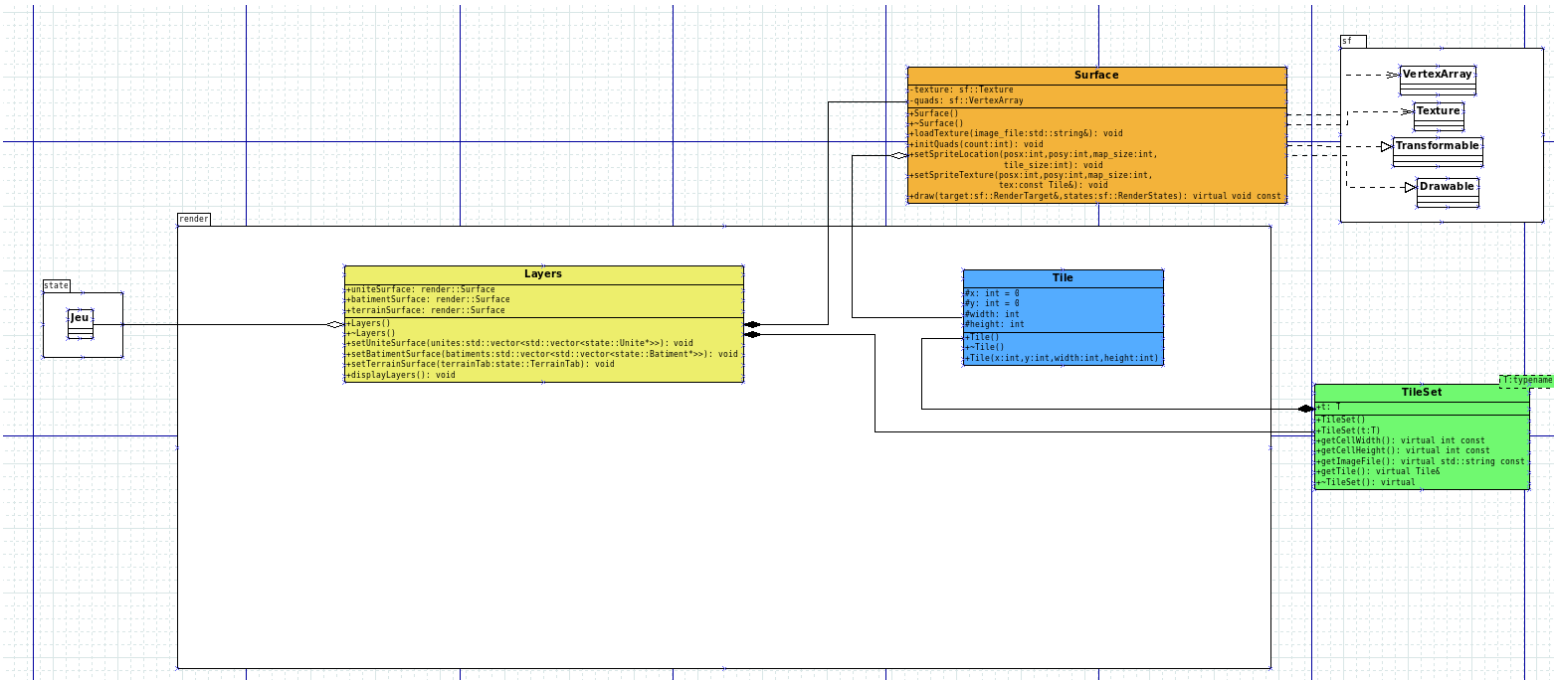

FIGURE 6 – Diagramme des classes de rendu.