Projet Worms
3IS 2017 - 2018

ABOUOBAYD Sanaa
PHENIX Clara

Professeur encadrant :

Philippe-Henri GOSSELIN

Figure 1 - Exemple du jeu Worms

ABOUOBAYD Sanaa
PHENIX Clara
Projet 3IS

Sommaire

ABOUOBAYD Sanaa
PHENIX Clara
Projet 3IS

1. Présentation générale

1.1 Archétype

L'objectif de ce projet est la réalisation du jeu Worms en 2D, avec les règles les plus simples. Un exemple est présenté en Figure 1.

1.2 Règles du jeu

Le jeu se déroule dans un unique monde en 2D et oppose deux équipes, une équipe verte et une équipe noire, composées chacune de trois personnages. Chacun son tour, les deux joueurs choisissent un de leurs personnages pour tirer sur ses adversaires. Un seul type d'armes est mis à disposition pour chaque personnage. Le joueur n'a droit qu'à un seul tir et à trois pas par tour. Un personnage meurt s'il est touché trois fois. Le premier joueur qui perd tous ses personnages a perdu.

1.3 Ressources

Figure 2 - Sprites pour les personnages

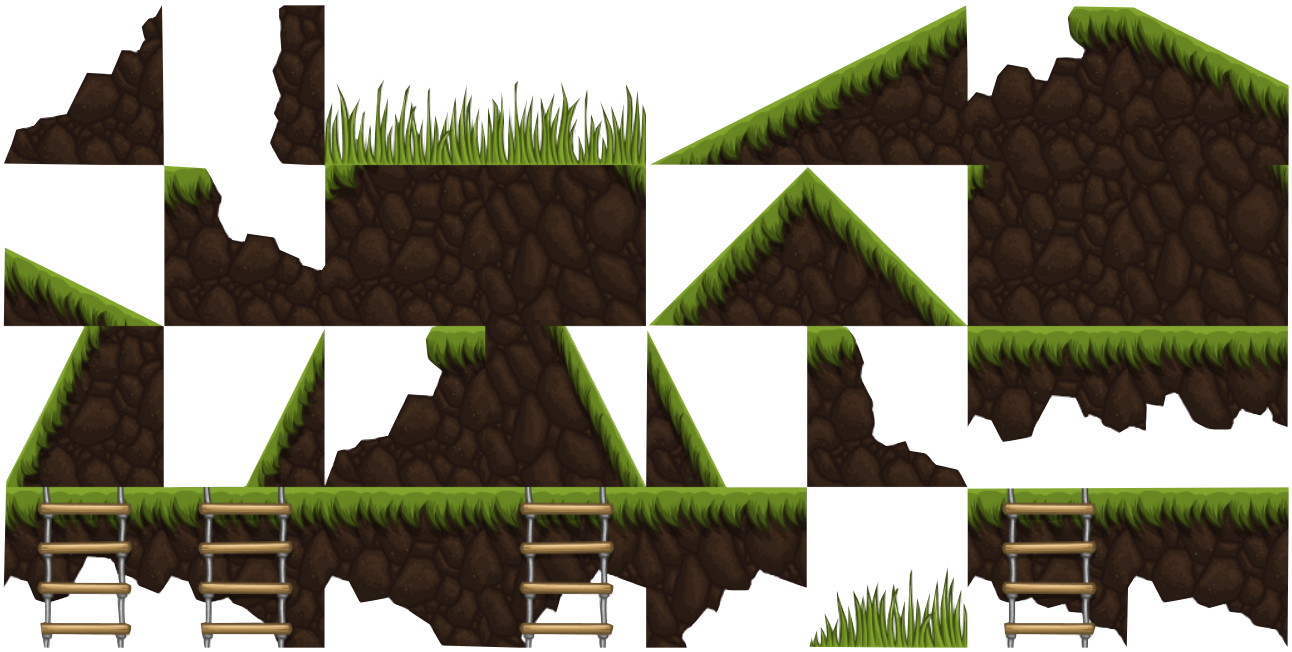

Figure 3 - Sprites pour la carte

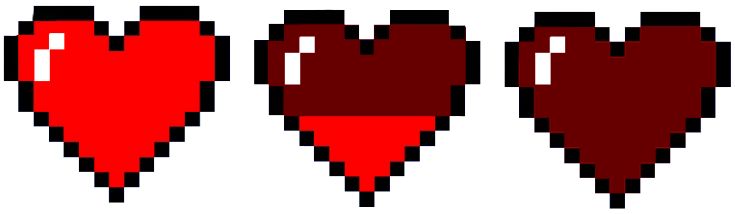

Figure 5 - Sprite pour les vies

2. Description et conception des états

2.1 Description des états

Un état du jeu est composé d'un ensemble d'éléments fixes (les cases), et des éléments mobiles (les personnages).
Les propriétés suivantes sont communes à tous les éléments:

- Coordonnées (x, y) dans la grille
- Identifiant de type d'élément : ce nombre indique la nature de l'élément (ie classe)

2.1.1 Etat éléments fixes :

La carte du jeu est composée de ''cases''. Ces cases sont elles aussi divisées en deux types :

- **Cases Sol :** ce sont les éléments infranchissables par les éléments mobiles et sur lesquelles les personnages peuvent se déplacer. Il est possible de les empiler pour rajouter du relief à la carte. Cet élément peut changer d'état au cours de la partie et devenir une case vide (définie plus loin). Le choix des textures qui constituent ces cases est purement esthétique et n'influence pas sur le déroulement du jeu.

- **Cases Espace :** ce sont les éléments franchissables par les éléments mobiles. On différencie deux types d'espace :
  - Les espaces *vides*
  - Les espaces *vies*, contiennent les vies accessibles pour les personnages et permettant le gain d'une vie pour le personnage qui le récupère.

2.1.2 Etat éléments mobiles :

Les éléments mobiles possède une direction (aucune, gauche, droite ou haut).

- **Personnage :** Cet élément est dirigé par le joueur, qui commande les propriétés de déplacement. Chaque personnage dispose d'un compteur de pas, qui limitera ses déplacements pendant un tour. Un personnage peut avoir deux états :
    - Etat *vivant* : C'est l'état normal, il peut tirer et se déplacer dès qu'il est sélectionné par le joueur.
    - Etat mort : Le personnage n'est plus visible à l'écran, lorsqu'il a perdu ses trois vies.

2.1.3 Etat général :

En plus des éléments décrits précédemment, le jeu possède les propriétés générales suivantes :

- **Epoque :** représente l'heure correspondant à l'état, ie le nombre de tic de l'horloge depuis le début de la partie.
- **Vitesse :** c'est la vitesse à laquelle l'état du jeu sera mis à jour.
- **Compteur de vies :** permet de compter le nombre de vies restantes que possède un personnage.
- **Compteur de parties gagnées :** le nombre de parties remportées par l'un des deux joueurs.

2.2 Conception logiciel

Nous avons représenté en figure 6 le diagramme UML, correspondant à la description des états.
Il en ressort deux groupes de classes principaux :

- Le groupe de classe Element. Ce groupe est construit à partir du polymorphisme. La classe abstraite Element est à la tête de cette hiérarchie. Toutes les classes décrivant un élément du jeu hérite de cette classe. Nous avons classé ces éléments en deux catégories : les éléments statiques et les éléments mobiles. Afin de les reconnaître nous avons donc créé une méthode isStatic(). En se basant sur le même principe, nous avons créé la méthode isSpace() pour la classe StaticElement.

- Le groupe de classe dédié au placement des éléments. Pour pouvoir situer ces éléments dans l'espace, nous avons réalisé la classe ElementTab qui représente une grille en deux dimensions. Chaque « Element » appartient à la grille ElementTab. La classe State permet d'obtenir les informations concernant l'état.

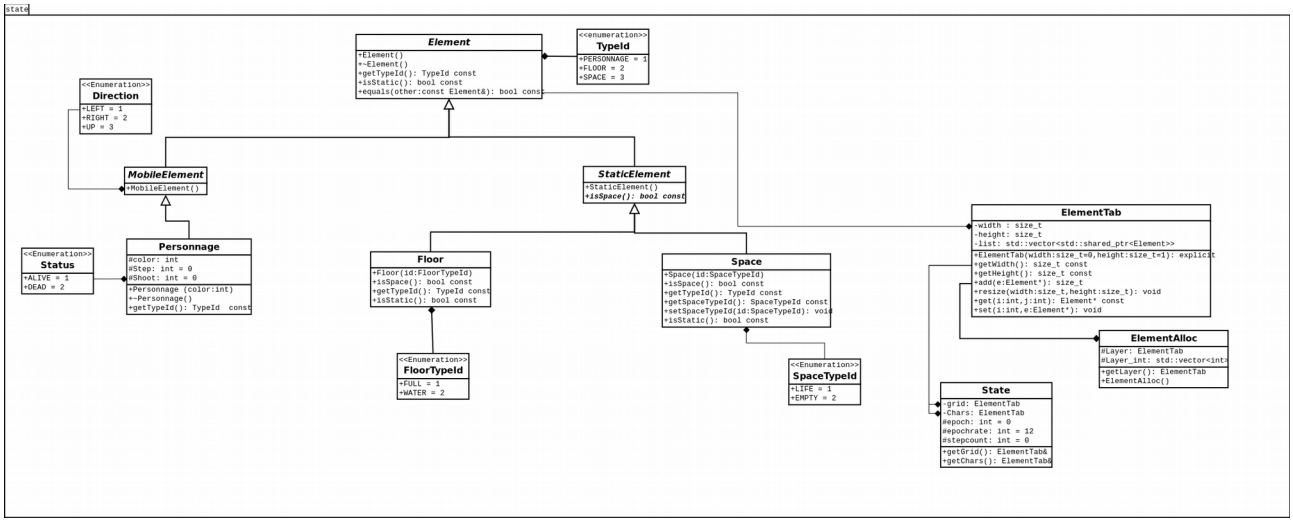

Figure 6 - Diagramme UML des états

3. Rendu : Stratégie et Conception

3.1 Stratégie de rendu d'un état

Nous avons choisi séparer notre carte en plusieurs couches (layers) : une couche background, une couche mer, une couche terre et par la suite une couche personnage et une couche information/bonus. Chaque couche correspond à un fichier .txt qui sera lu et qui permettra d'afficher la carte.
Chaque élément du fichier correspond à la position de la tiled équivalente, dans le sprite. Grâce à cette information on va pouvoir associer chaque élément à un sprite et ainsi pouvoir l'afficher graphiquement.

3.2 Conception logiciel

Plans : La classe Layer sert à donner des informations 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.

Surface : Une surface contient une texture du plan et une liste de paire de paires de quadruplets de vecteurs 2D. Chaque quadruplet possède des éléments *texCoords*, qui correspondent aux coordonnées des quatre coins de la tuile à sélectionner dans la texture, et des éléments *position*, qui correspondent aux coordonnées des quatre coins du carré où doit être dessiné la tuile à l'écran.

ABOUOBAYD Sanaa
PHENIX Clara
Projet 3IS

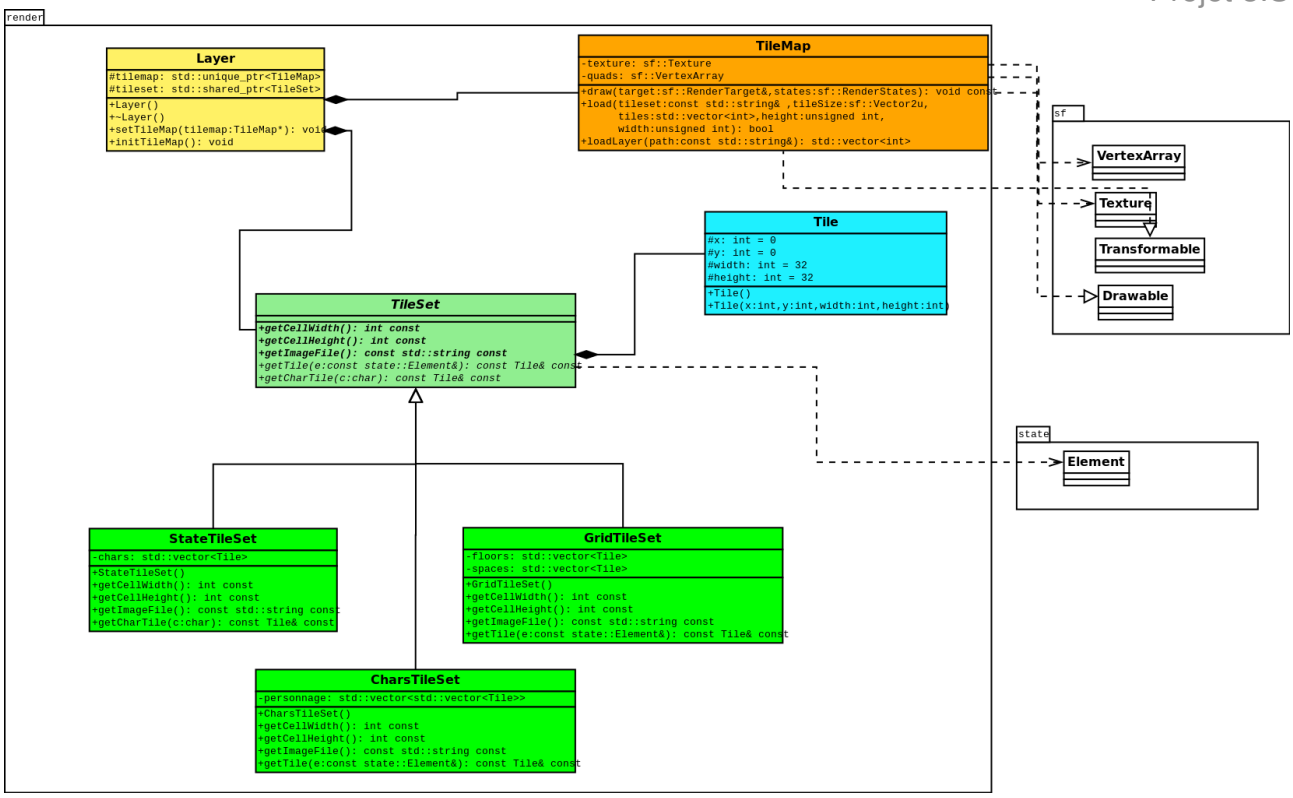

Figure 7 - Diagramme UML du rendu