Etude du format TMX

# But du document

Parvenir a décrypter le fonctionnement et les subtilitées du format TMX afin de pouvoir implementer un parser pour charger des maps créées via le logiciel de mapping Tiled qui utilise en interne le format TMX.

# Pourquoi TMX

Le TMX (Tile Map XML) semble être le format le mieux reconnu dans le domaine du tile mapping. Notre démo de RPG pour l’EIP se basant justement sur du tile mapping il est intéressant de partir sur un format normalisé.

# Avantage de TMX

TMX permet de décrire des cartes faites à partir de tiles. Ces tiles peuvent être de toutes tailles. Il peut il y avoir autant de layers que bon vous semble, autant de tiles set que vous le souhaitez, TMX permet aussi de gérer les propriétées sur la plupart des éléments. TMX permet aussi de former des groupes d’objets que l’on peu manipuler.

# Désavantage de TMX

Ce format ne gère pas les animations ce qui est un gros bémol car il peut arriver que l’on souhaite faire un tile animé, par exemple de l’eau ou une porte qui s’ouvre. Le format TMX ne permet pas cela et aucun de ses homologues ne parvient à le gérer sans perdre des fonctionnalités inhérentes au tile mapping. Il faut donc prévoir des subterfuges permettant de pallier à cet écueil.

# Structure d’un fichier TMX

Les fichiers TMX ont une structure faite d’XML. Voici les principales balises XML permettant de structurer un fichier TMX :

* Map
* Tileset
  + Tileoffset
  + Image
  + Terraintypes
  + Terrain
  + Tile
* Layer
  + Data
  + Tile
* Objectgroup
  + Object
  + Ellipse
  + Polygon
  + Polyline
* Imagelayer
* Properties
  + Property

Je vais passer en revue chacune de ces balises afin d’expliquer son rôle et son fonctionnement. Et pas à pas expliquer les relations entre chacunes d’elles.

# Balise Map

La balise map permet de définir les propriétées globales de la map. A savoir :

* La version / version (version du format TMX, généralement 1.0)
* L’orientation / orientation (« orthogonal », « isometric » ou « staggered »)
* La largeur / width (La largeur de la map en tiles)
* La hauteur / height (la hauteur de la map en tiles)
* La largeur d’un tile / tilewidth (la largeur d’un tile en pixels)
* La hauteur d’un tile / tileheight (la hauteur d’un tile en pixels)
* La couleur d’arriere fond / backgroundcolor (la couleur d’arrière fond de la map)

# Balise tileset

La balise tileset permet de gérer des « patrons » de tiles. Chaque balise tileset possède plusieurs propriétées à savoir :

* Identifiant global / firstgid (Il s’agit tout simplement d’un offset pour rendre unique l’identifiant de chaque tile appartenant au tileset).
* source (il s’agit du nom du fichier .tsx (tile set xml) contenant la description de chaque modèle de tile utilisable dans la map)
* name (il s’agit du nom sous lequel sera répertorié le tileset dans la balise map)
* tilewidth (c’est la largeur maximale en pixel d’un tile)
* tileheight (c’est la hauteur maximale en pixel d’un tile)
* spacing (c’est l’espace en pixels entre chaque tiles dans le tileset)
* margin (c’est la marge entourant chaque tile contenu dans le tileset)

## Balise tileoffset

La balise tileoffset permet de specifier un offset pour chaque tile faisant partie du tileset. La balise tileoffset possède deux proprietées :

* x (Qui est un offset horizontal)
* y (Qui est un offset vertical)

## Balise image

La balise image permet de definir l’image dans laquelle sont découpés les tiles définis dans la balise tileset. La balise image peut contenir plusieurs attributs :

* source (qui definit le chemin du fichier qui sert de modele d’image)
* trans (qui definit une keycolor de transparence par exemple 00FF00 pour du vert)
* width (qui definit la largeur de l’image en pixels)
* height (qui definit la height d’une image en pixels)

Notez bien que chaque tileset ne doit posséder qu’une seule balise image.

## Balise terraintypes

Cette balise permet de définir un tableau de terrains. Les tiles qui sont définis dans le tileset utilisent d’ailleurs ces terrain pour définir a quel type de terrain ils appartiennent.

### Balise terrain

Cette balise permet de définir un terrain appartenant a la balise terraintypes. Chaque terrain possède deux attributs :

* name (qui représente le nom du terrain.)
* tile (qui est en fait l’id du tile qui représente le terrain visuellement)

## Balise tile

La balise tile permet de définir un modèle de tile dans la balise tileset. Chaque balise tile peut posséder plusieurs attributs :

* id (l’id local dans le tileset du tile)
* terrain (cet attribut est un peu particulier, il permet de déterminer de quels terrains est fait le tile, pour ce faire on définit le terrain composant les 4 coins du tile (haut-gauche, haut-droit, bas-gauche, bas-droite) via les identifiants des terrain séparés par des virgules).
* Probability (probabilité qu’a le tile de se mettre à un endroit donc les tiles adjacents répondent au prédicat terrain (lorsque les 4 coins matches)).

# Balise Layer

La balise Layer permet de définir une « couche » de tiles contenus dans la balise map. Chaque Layer possède plusieurs attributs :

* name (Le nom du layer)
* x (la position x du Layer en tiles, par défaut x vaux 0)
* y (la position y du Layer en tiles, par défaut y vaux 0)
* width (la largeur du layer en tiles)
* height (la hauteur du layer en tiles)
* opacity (l’opacité du layer, 0 rend le layer totalement transparent, a l’inverse 1 le rend totalement opaque).
* visible (définit si le layer est visible ou non, peut être très utile pour faire des masques de collisions.). 1 rend le layer visible, 0 le rend invisible.

## Balise Data

La balise data permet définir les données les données du layer, a savoir la définition des tiles composant le layer. La balise data peut comporter deux attributs :

* encoding (il s’agit du mode d’encodage des données du Layer, si cet attribut est définit il peut valoir « base64 » ou « csv », en revanche si il n’est pas définit, cela signifie que les données des tiles sont exprimées en XML)
* compression (il s’agit du mode de compression utilisé pour compresser les données des tiles. Cet attribut peut valoir « gzip » ou « zlib »).

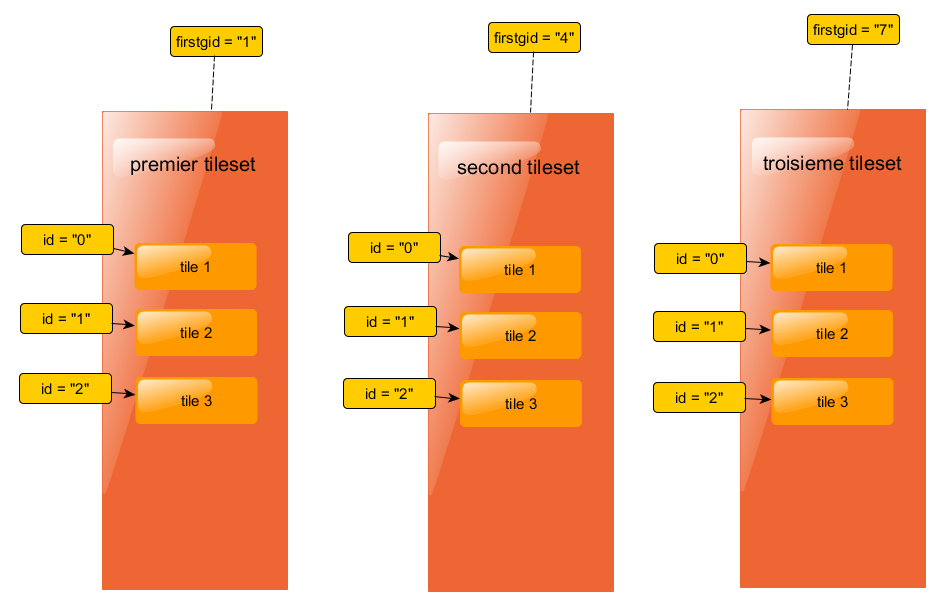
## Balise Tile

Attention à ne pas confondre cette balise tile avec celle contenue dans le tileset. La balise tile ci-décrite permet de définir un tile contenu dans le layer qui se BASE sur un MODELE de tile qui lui est définit dans un des tileset. Chaque balise tile possède un attribut :

* gid (il s’agit du global identificateur correspondant à un des tiles définit dans un des tileset.)

Souvenez-vous, chaque tileset possède un attribut firstgid, cet attribut jouait le rôle d’offset, imaginons que vous ayez 3 tileset différents, chacun d’eux définit 3 tiles. Le premier tileset à un firstgid de « 1 », le second tileset aura un firstgid de « 4 », enfin, le troisième aura un firstgid de « 8 ».

Pour y voir plus clair voici un schéma récapitulant ce que je viens de définir :



Vous pouvez remarquer que chaque tile possède un identifiant qui n’est pas unique, ce qui va permettre de rendre unique chaque tile faisant partie d’un tileset, c’est le firstgid du tileset qui joue le rôle d’offset. Ainsi, les véritables valeurs des id de chaque tile ne seront pas 0, 1, 2, 0, 1, 2, 0, 1, 2. Mais 1,2,3,4,5,6,7,8,9. Ceci est dû au fait que chaque identifiant de tile se voit ajouté le firstgid de son tileset.

Ainsi, définir un tile dans votre balise <data> avec un gid (global id) de 5 correspondra au tile 2 définit dans le tileset donc le firstgid vaux « 4 ».

# Balise objectgroup

La balise objectgroup fonctionne de la même manière qu’un layer hormis le fait qu’elle comporte uniquement des balises object en son sein. La balise objectgroup permet donc de faire des agrégats d’objets allant du polygon en passant par l’ellipse ou encore la polyline. Ce peut être très utile pour définir des masques de collisions très fins. Vous pouvez aussi bien entendu rendre l’objectgroup invisible ou lui affecter une couleur spécifique. La balise objectgroup peut comporter plusieurs propriétées :

* name (représente le nom du groupe d’objets)
* color (représente la couleur utilisée pour l’affichage des objets de ce groupe)
* x (représente la coordonnée x du groupe en tiles)
* y (représente la coordonnée y du groupe en tiles)
* width (represente la largeur du groupe d’objets en tiles (potentiellement inutile))
* height (represente la hauteur du groupe d’objets en tiles (potentiellement inutile))
* opacity (représente l’opacité du groupe d’objet, 0 le groupe est totalement transparent, 1 le groupe est totalement opaque).
* Visible (représente la visibilité du groupe d’objets, 1 le groupe d’objets est visible, 0 il est invisible.)

La balise objectgroup peut contenir des balises properties ainsi que des balises object.

## Balise object

La balise object permet de définir des objets qui ne seront pas forcément alignés comme les tiles, c’est-à-dire que les balises <object> peuvent avoir une position quelquonque dans leur balise <objectgroup>. Ça peut être utile pour définir des zones spécifiques, comme des zones de spawn, de warp, de sortie. La balise object peut avoir plusieurs attributs :

* name (le nom de l’objet, il s’agit d’une chaine de caractères arbitraire)
* type (le type de l’objet, il s’agit d’une chaine de caractères arbitraire)
* x (la coordonnée x de l’objet dans le objectgroup)
* y (la coordonnée y de l’objet dans le objectgroup)
* width (la largeur de l’objet en pixels, par défaut cette propriété doit valoir 0)
* height (la hauteur de l’objet en pixels, par défaut cette propriété doit valoir 0)
* gid (une référence vers un tile, c’est optionnel)
* visible (permet de définir si l’objet est visible ou non)

La balise Layer est très utile pour tout ce qui est répétitivement placé et aligné sur une grille de tiles. Cependant il peut arriver que vous souhaitiez être libre dans le placement de certains objets, c’est pour cela que la balise object existe.

## Balise ellipse

La balise ellipse permet de marquer le contenu d’un objet comme étant une ellipse. La balise ellipse peut se voir affecter plusieurs attributs :

* x (coordonnée d’abscisse locale à la balise object permettant de placer le point central de l’ellipse (généralement ignoré par l’éditeur))
* y (coordonnée d’ordonné locale à la balise object permettant de place le point central de l’ellipse (généralement ignoré par l’éditeur))
* width (largeur de l’ellipse exprimée en pixels (généralement ignoré par l’éditeur))
* height (hauteur de l’ellipse exprimée en pixels (généralement ignoré par l’éditeur))

## Balise polyline

La balise polyline permet de définir une ligne brisée à partir d’une suite de points. La balise polyline prend un attribut :

* points : qui sont une suite de coordonnées 2d définissant un point dans le plan de l’objectgroup. Par exemple, si notre ligne brisée doit être composée de 3 points [4 ;5], [6 ;2],[9 ;4], alors l’attribut points aura la valeur suivante : points=’’4,5,6,2,9,4’’.

# Balise imagelayer

La balise imagelayer permet de définir un layer qui n’est composé que d’une image. Ainsi cela permet de se servir d’une image de background sans avoir besoin de faire de tile mapping si ce n’est pas nécessaire. Chaque balise imagelayer peut comporter des balises properties et une balise image.

# Balise properties

La balise properties est un conteneur de propriétés. Elle peut être contenue par une balise map, tile (appartenant à un tileset), a un layer, ou encore à un object.

## Balise property

La balise property représente une propriété. Chaque balise property peut avoir deux attributs :

* name (qui est le nom de la propriété)
* value (qui est la valeur de la propriété)

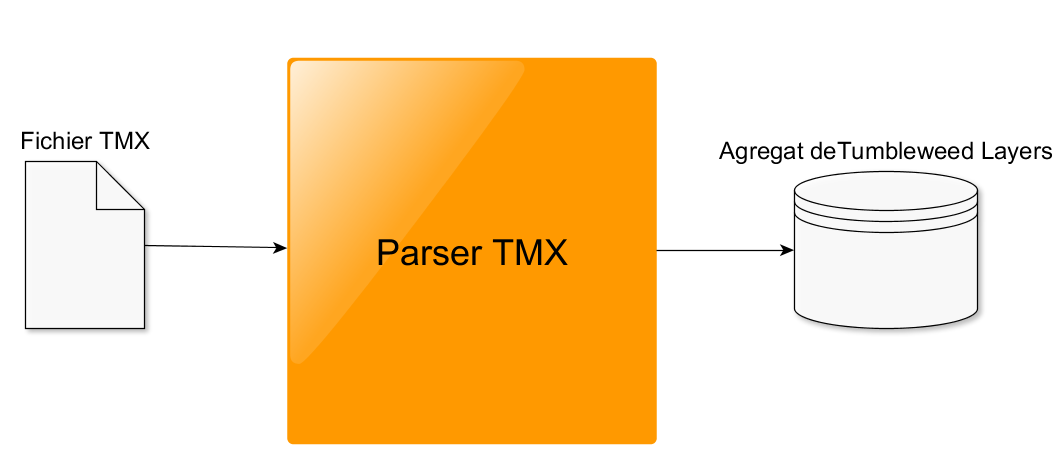
# Construire un parser TMX

Je vais maintenant vous accompagner tout le long de la création d’un parser TMX. Notre parser sera capable en l’associant au framework javascript tumbleweed de créer plusieurs layers et de les garnir avec des sprites correspondant aux tiles décrit dans le fichier TMX que vous aurez parsé.

La première chose à faire est d’avoir une approche top-down de ce que l’on souhaite réaliser, en partant d’un use case simple puisque unique. Tout ce que nous souhaitons c’est que l’utilisateur de notre parser puisse charger un fichier TMX afin d’obtenir en sortie des layers configurés selon le fichier TMX.

Pour se faire nous allons créer une classe TMXParser, qui prend en entrée le fichier TMX à charger et fourni en sortie un layer qui représente la map chargée. Si le parsing échoue on renverra null.

Voici donc le use case :



En interne notre parser TMX va avoir besoin de plusieurs composant logiciels lui permettant d’analyser le contenu du fichier TMX fournit en entrée. C’est ce qui va permettre à notre parser de générer des sprites et des layers de manière adéquate.

Analysons plus en profondeur la transposition qui doit être faite pour passer de la description TMX à la structure en Layers de tumbleweed.

De base lorsqu’on charge un objet XML il est automatiquement transformé en objet javascript. Cela permet de pouvoir le manipuler simplement. Tout le parsing est donc fait pour nous. Ce qu’il faut faire, c’est charger en mémoire les images utilisées par cet objet XML.

Pour générer nos tumbleweed Layers nous allons parcourir chaque Layer de notre object TMX créer un tumbleweed Layer équivalent puis le remplir avec des sprite qui seront fabriqués à partir des spécifications fournies dans la balise data et dans les tilesets.

On a donc un layer qui servira de racine. Son seul but est de contenir tous les layers afin que ces derniers soient dessinés dans l’ordre.