Projet OUAW

Figure 1 - Logo du jeu

Léo GIMENEZ

Raphaël PICHON

Luc WERNER

Charlie XANH

Table des matières

1- Présentation

Dans le cadre de notre Projet Logiciel Transversal, il nous est demandé de réaliser un jeu vidéo tour par tour permettant d'exploiter les modules étudiés au cours de l'année : Génie Logiciel, Algorithmique, Programmation Parallèle et Web Services.
Notre groupe a décidé de baser son jeu sur ***For the King*** qui est un RPG -Roguelite alternant déplacement et combat, les deux en tour par tour.

1.1 Définition de notre jeu

RPG - "Role Play Game" : Traduit en "Jeu de Rôle", cela définit un genre de jeu vidéo où les joueurs incarnent un ou plusieurs personnages qui évoluent au fil d'une quête.

Roguelite : Ce terme définit un *sous-genre* du jeu vidéo. Ce sous-genre est dérivé du terme *Roguelike*, qui définit un jeu vidéo avec les caractéristiques suivantes :

- Une mort permanente : le personnage joué n'est pas sauvegardé d'une partie à l'autre.
- Des niveaux procéduraux : les niveaux sont générés de façon procédurale.
- Des combats tour par tour.
- Pas de fin.

Dans notre cas plusieurs de ces caractéristiques ne sont pas respectées, comme par exemple **les niveaux procéduraux** ou **l'absence de fin**, ce qui fait de notre jeu un *Roguelite*.

1.2 Règles de notre jeu

Nous avons choisi d'appeler notre jeu ***OUAW : "Once Upon A WEI"***, les joueurs y incarnent des étudiants fictifs de l'ENSEA qui affrontent des épreuves dans le but de valider leurs années et retrouver le WEI perdu depuis des temps immémoriaux. Les règles sont les suivantes :

- n-joueurs jouent chacun un élève au tour par tour
- A chaque tour le groupe lance un dé pour se déplacer
- A chaque étape des PO est donnée comme monnaie
- Possibilité de récupérer de l'équipement à la K-fet en échange de PO –
- Le groupe perd lorsqu'il ne valide pas son année
- Chaque joueur à des caractéristiques et des équipements qui lui permettent de passer les épreuves lors d'une phase de combat.

Nous n'avons pas, pour le moment, définie un système d'équilibre concernant les caractéristiques, les dégâts, etc… Cependant les classes sont prévues pour recevoir un tel équilibrage.

1.3 Ressources

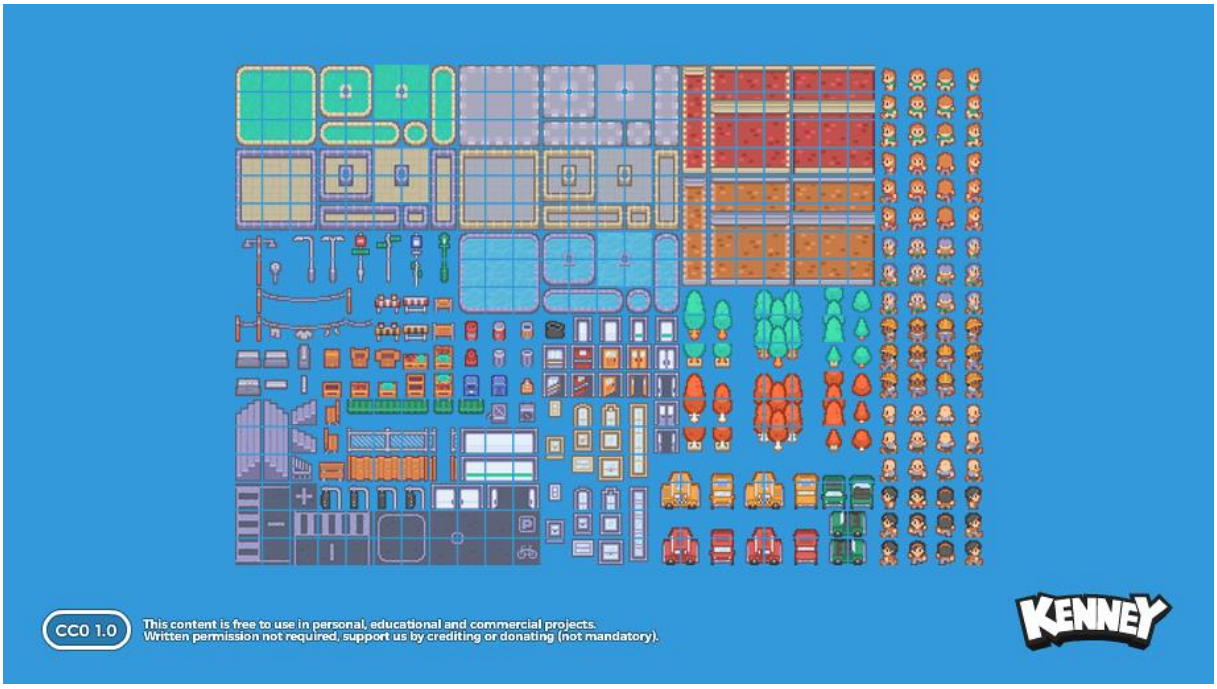


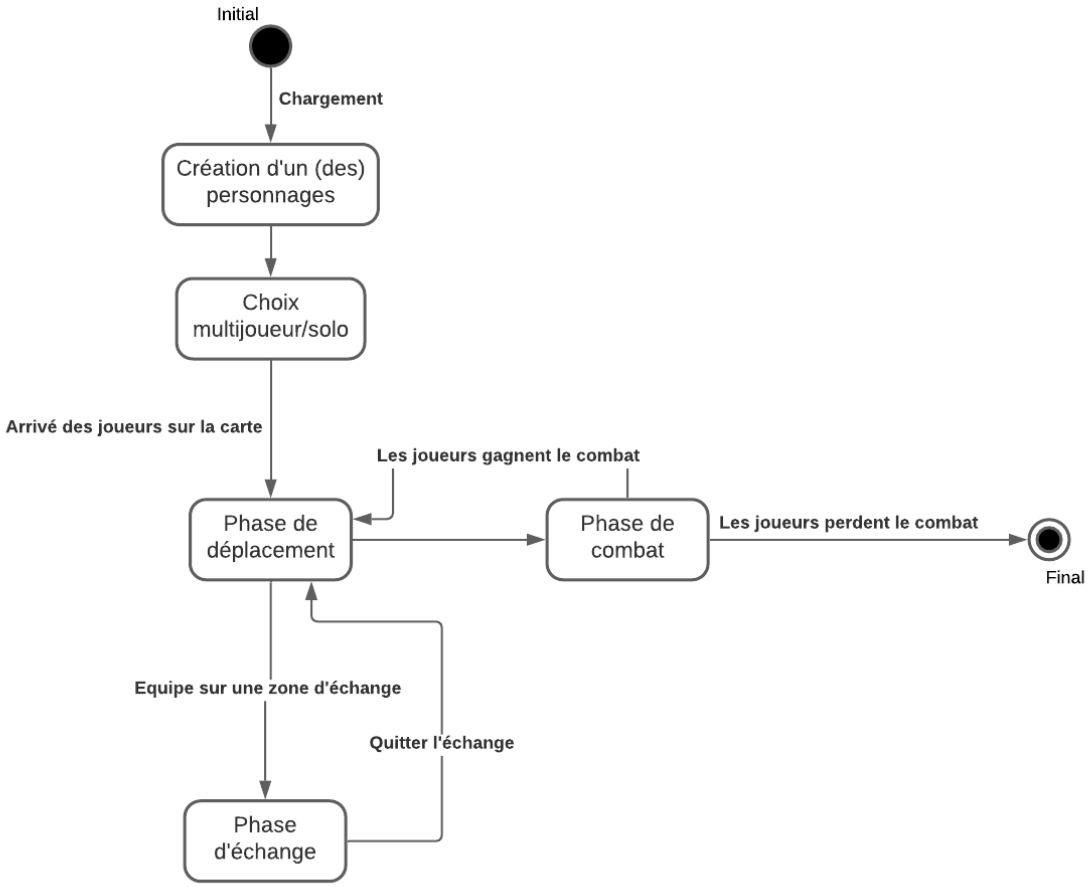
Figure 2 - Textures pour le jeu

Source : https://kenney.nl/assets/rpg-urban-pack

2- Description et états

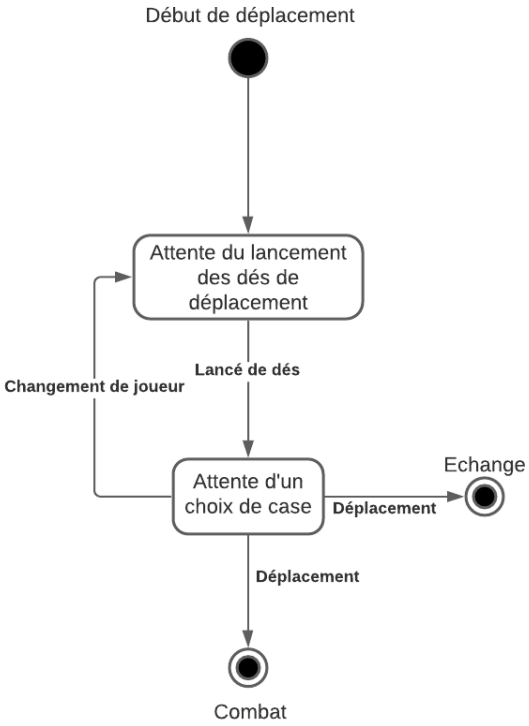
La partie se déroule donc lors de 4 phases de jeu, elles sont décrites dans les diagrammes d'états ci-dessous :

2.1 Déroulement global d'une partie



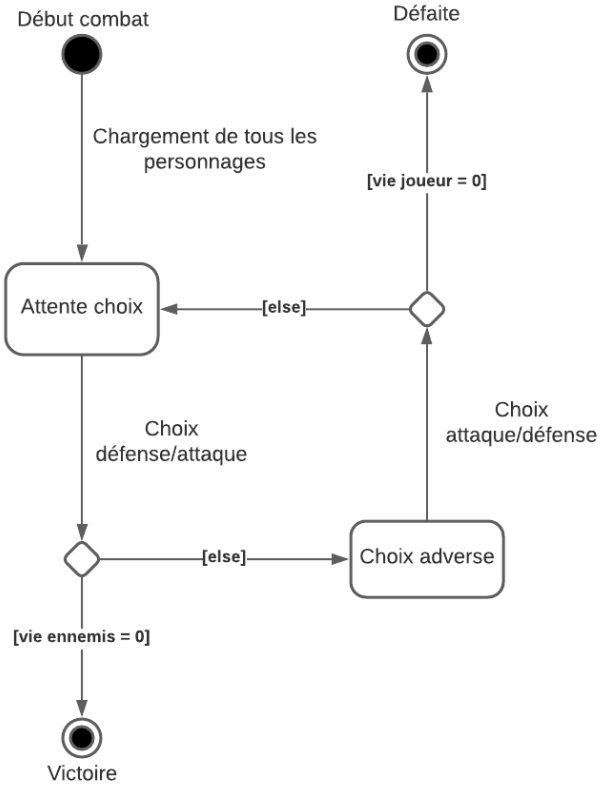
La partie commence par deux menus permettant de choisir son personnage et de jouer en multi joueurs. Puis la partie tourne autour de trois phases, celle de déplacement, de combat et d'échange. On note que la partie se termine uniquement lors d'une phase de combat.

2.2 Phase de déplacement



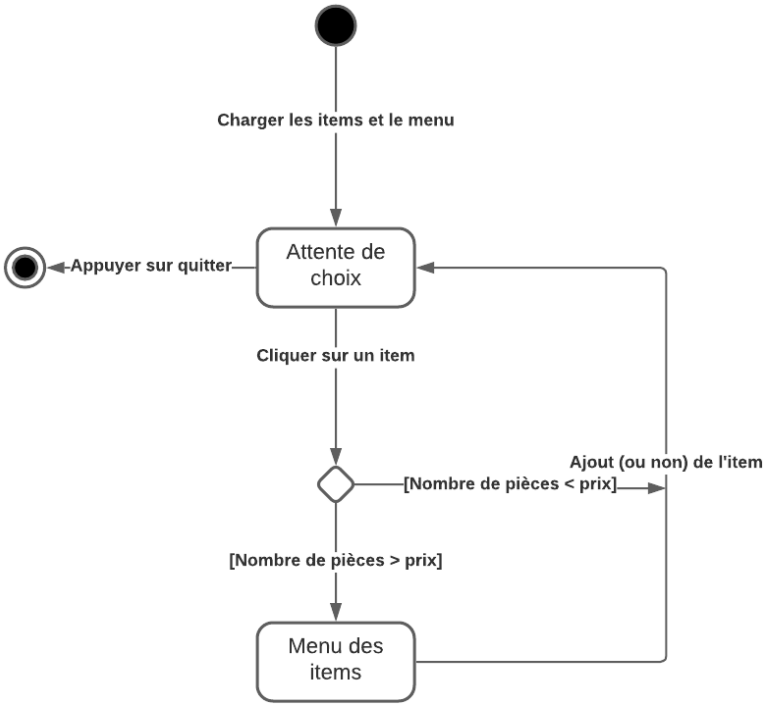
La phase de déplacement tourne en boucle tant que le groupe n'est pas arrivé à une phase de combat ou une phase d'échange.

2.3 Phase de combat



La phase de combat est une phase classique de combat au tour par tour. Tant que le joueur ou l'adversaire est en état de combattre, le combat continu.

2.4 Phase d’échange



La phase d’échange permet d’échanger de l’argent contre des items.
Cette phase se termine quand le joueur le décide.

3- Conception Logiciel

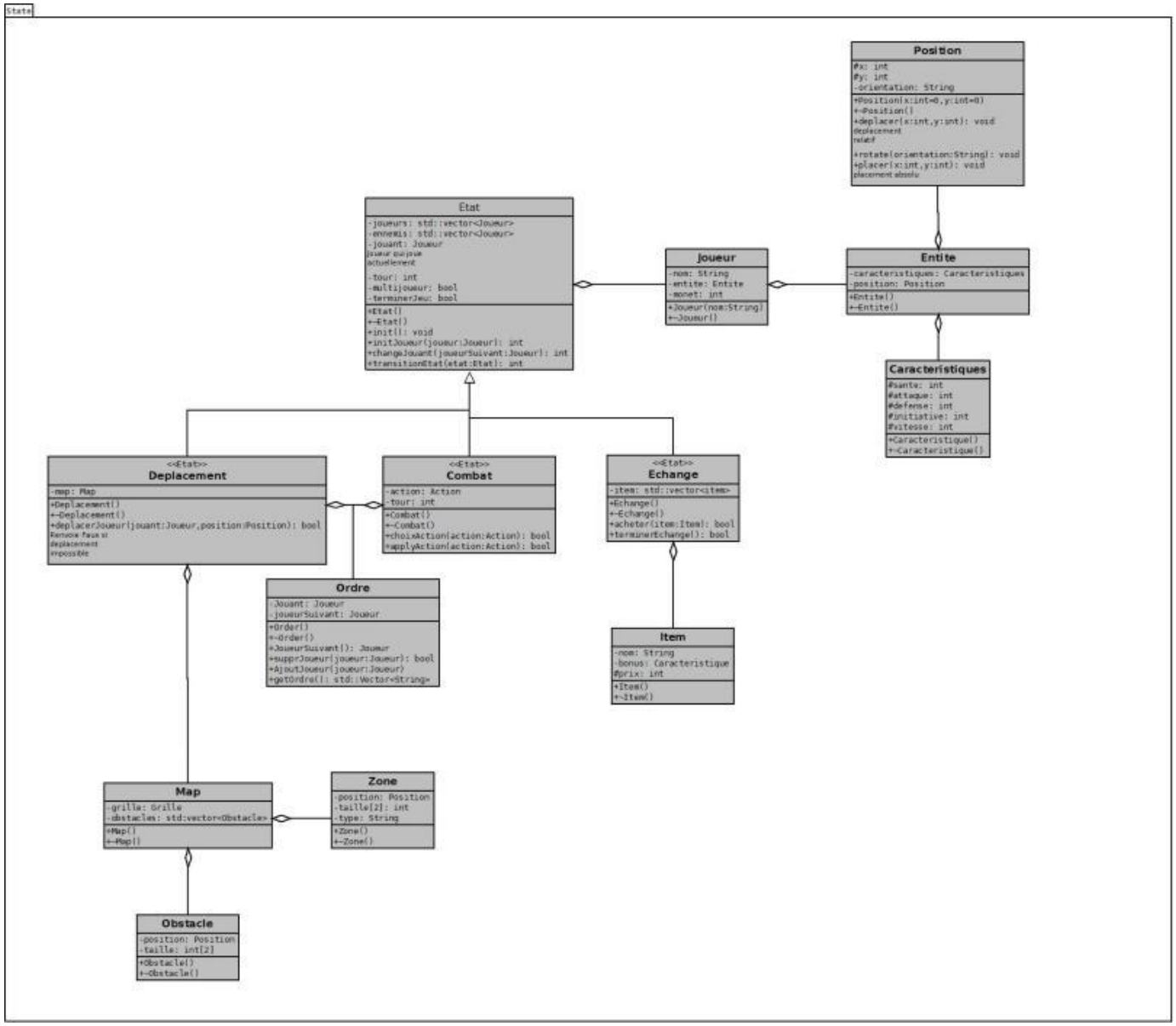


Figure 3 - Diagramme d'état

4- Rendu : Stratégie et conception

4.1 Stratégie de rendu

Le rendu correspond à la partie qui va gérer l'aspect graphique du jeu. Nous devons récupérer l'état de la partie et afficher les informations nécessaires à l'écran.

Nous utiliserons la librairie SMFL2.0 pour toute la partie rendue.

La stratégie de rendu est de découper une scène à afficher en plusieurs plans superposés. La stratégie reste identique suivant la situation de l'état (map, échange ou combat). Nous allons afficher un ensemble de textures (Tuiles) suivant une certaine position appartenant toutes à un plan (layer).

Le dernier plan à afficher sera celui du décor de fond stocké (ou non) sous la forme d'un fichier texte suivant l'état, chargé et mis à jour au centre du joueur. Le second plan est celui des personnages. Le premier plan est celui des menus et interactions de l'utilisateur.

Pour le dernier plan :

- Lors d'un déplacement sur la carte, la stratégie est d'observer la position du joueur sur la carte et de centrer l'affichage en conséquence. Le fond ne changera que lorsque le joueur se déplace. Pour une situation de combat ou d'échange, le fond est fixe et sera inchangé.

Pour le second plan :

- Les personnages seront affichés directement sur la surface précédente sous forme de sprites. Une animation sera possible pour rendre le rendu plus dynamique lorsque le joueur doit prendre une décision. Une fréquence de rafraichissement des personnages (environ 30Hz) suivant une routine de sprites à afficher sera mis en place.

Pour le premier plan :

- Le premier plan est l'ensemble des menus avec lesquels l'utilisateur peut interagir. Ils seront donc rafraîchis si l'utilisateur souhaite effectuer une action. Les textures seront donc modifiées suivant la position de la sourie et de son interaction avec l'interface.

4.2 Conception du rendu

Voir figure … pour le die de rendu

Le cœur de notre rendu se situe autour des layer, chaque layer (3) est spécialisé dans son rendu et possède des attributs uniques. Par exemple le layer charger de la map n'aura que des tiles de type MapTile. Chaque tile possède une texture et une position. Ainsi à l'aide d'un appel successif sur l'ensemble des textures des tiles appartenant aux layers, il est possible de créer la surface finale à afficher lors d'un appel de getScene(). Les différentes classes sont les suivantes :

Scene :

Contient la boucle d'affichage et les textures finales qui seront affiché à l'écran. La Scene vient synthétiser la partie rendue en proposant d'actualiser la texture finale à l'aide de setLayer() et getLayer. La fonction draw scene vient donc afficher les textures qui ont été actualisées.

Layer :

Contient un ensemble de tiles, chaque layer correspond à une couche à afficher et sera synthétisé sous forme d'une texture par Scene. getLayer() permet d'obtenir la surface associée à l'ensemble des tiles qu'elle contient.

Tile :

Un tile est un élément de layer qui possède des coordonnées pour être positionné sur l'écran. Ainsi pour l'afficher la carte une matrice de tile sera nécessaire. Seul un vecteur de tile sera nécessaire pour les Player et les Menus. Ces Tile pouvant posséder des animations, un vecteur de texture est disponible dans les menus et les players. Ainsi chaque Texture de ce vecteur correspond à une orientation pour notre personnage, ou à une interaction avec l'utilisateur.

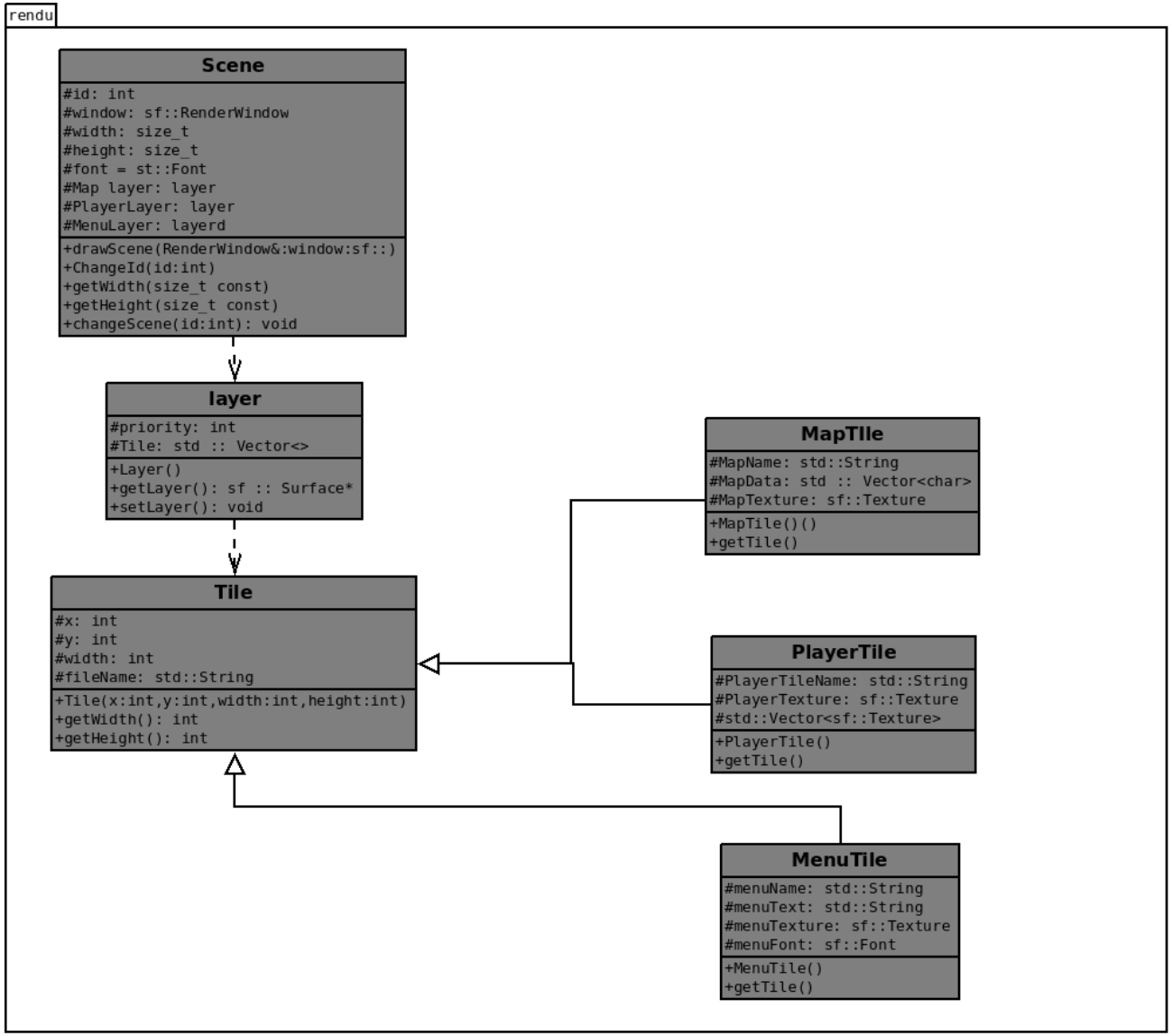


Figure 4 - Diagramme de rendu