Projet royal war

David Kha, Mazhou HU, Florian Palmier, Fredrick Omgba-Abenah

Figure 1-Exemple du jeu que nous voudrions faire

Présentation Générale **2**
- Archétype 2
- Règles 2
- Textures 3
  - Terrain 3
  - Décor 3
  - Nature 4
  - Projectile 4
  - Bâtiment 5
  - Personnages 6
  - Résultat 7

Présentation Générale

Archétype

L’objectif de ce projet est de créer un jeu comme Advanced War avec nos propres règles.

Règles

- Jeu en 1 vs 1, et contre l’IA
- Chaque personne possède une base et a la possibilité de faire spawn des unités différentes : soldat, lancier, cavalier, archer, dragon, mage et ballista, . Chaque personnage a un coût différent pour le faire spawner, a un nombre de mouvement limité suivant le terrain et a des dégâts différents
- La map est parsemée de villages, chaque village génère 100 or. Plus le joueur contrôle de village, plus il est facile de spawner une armée rapidement.
- Le but est de détruire la base de l'ennemi
- Chaque classe a ses propres caractéristiques

Textures

Le jeu se déroule sur une carte séparée par une grille dont les composants font 16x16 pixels. On trouvera ci-dessous les textures nécessaires au développement de la carte de jeu, qui se trouvent dans le dossier *./ res / texture*

Terrain

Figure 2- Texture pour les terrains

Décor

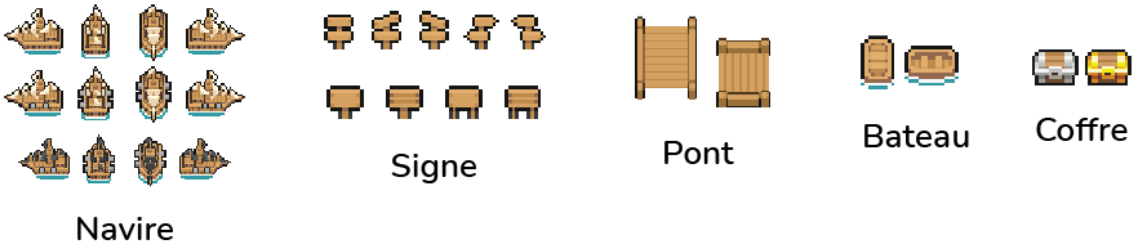

Figure 3 -Texture pour les décors

Nature

Figure 4 -Texture pour la nature

Projectile

Figure 4 -Texture pour la projectile

Bâtiment

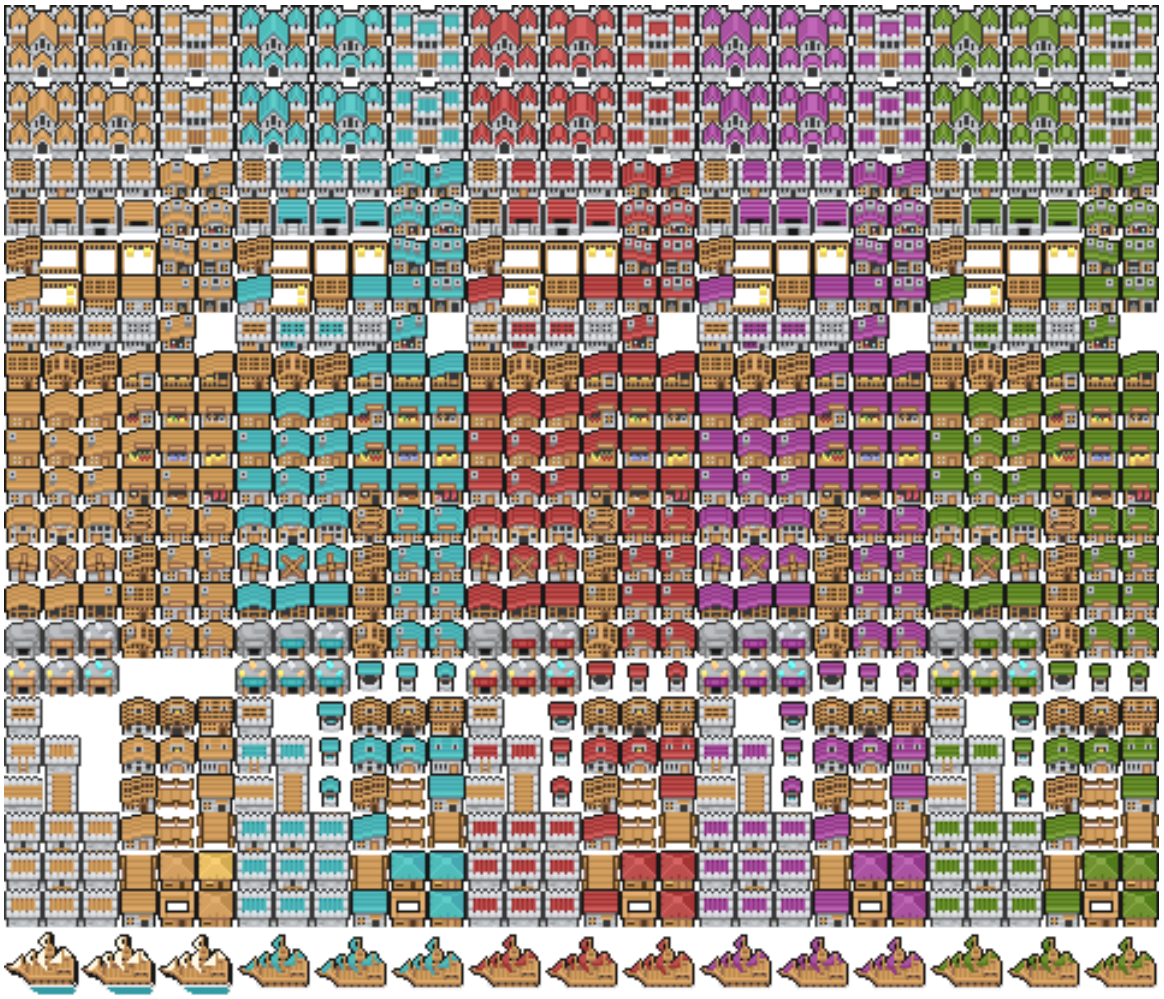

Figure 4 -Texture pour les batiements

Personnages

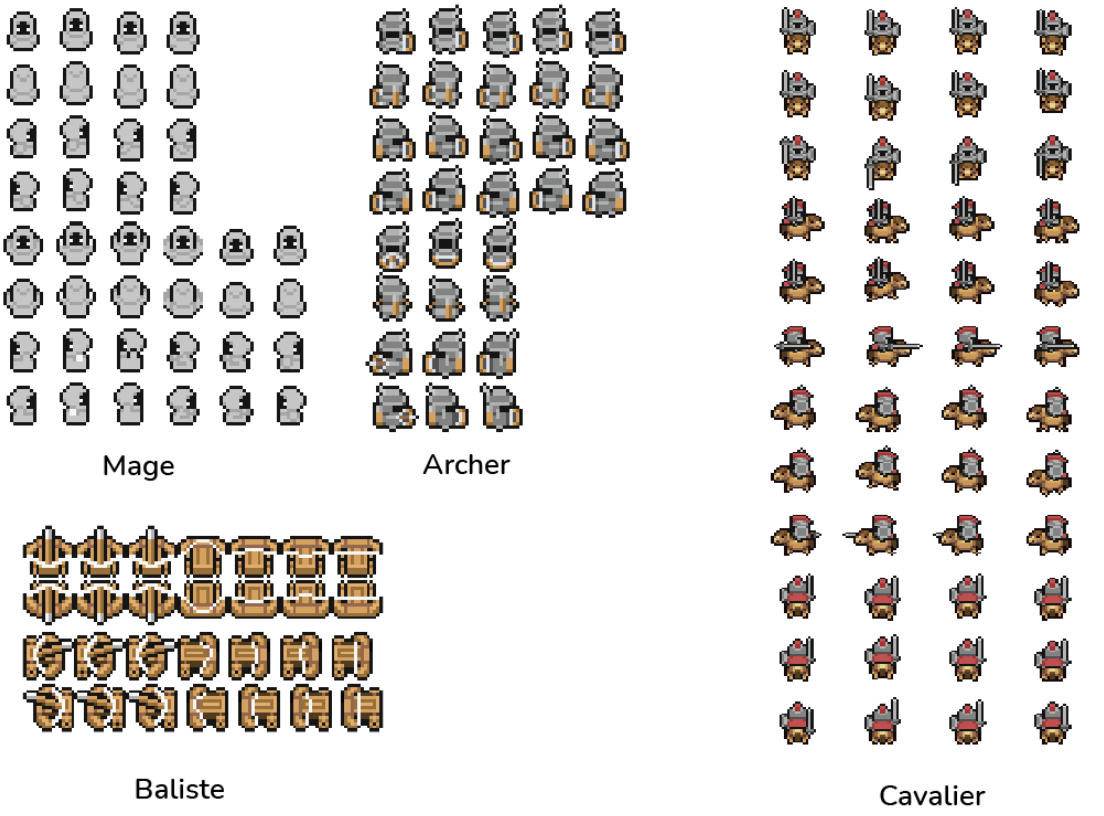

Figure 5.1 -Texture pour les personnage

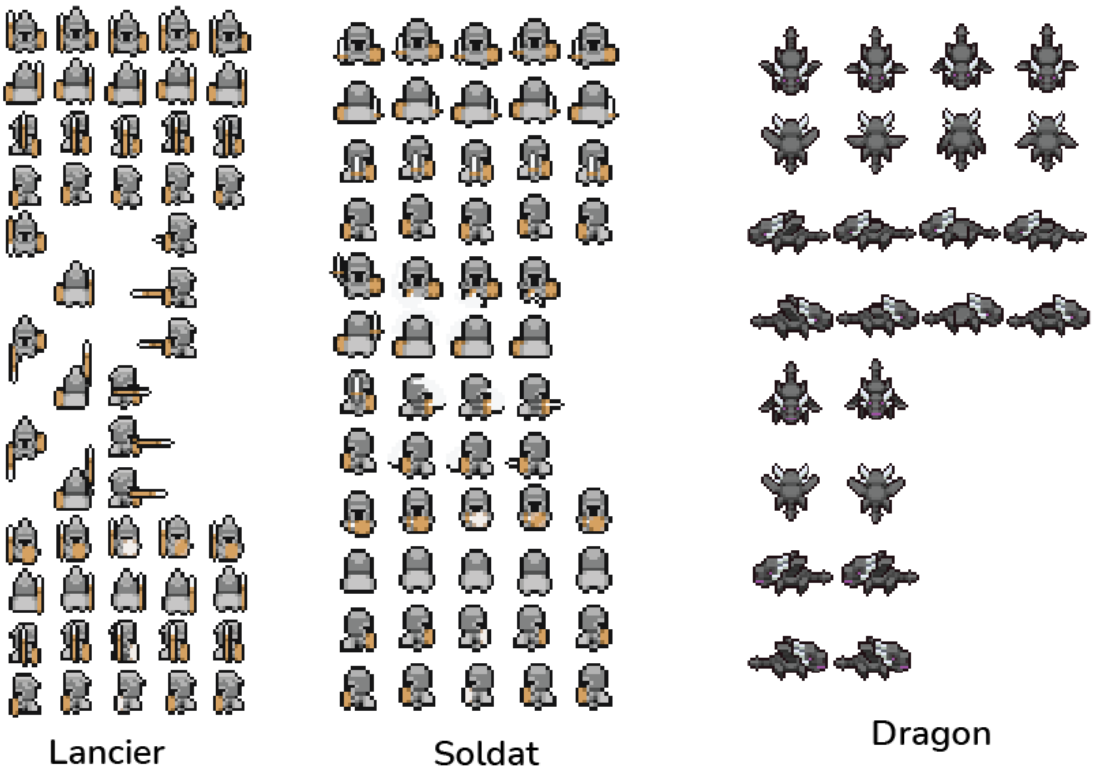

Figure 5.2 -Texture pour les personnages

Résultat

En utilisant ces ressources, on obtiendra une map qui ressemblera à l'image ci-contre.

Figure 6 -l'exemple de la carte créée

Description et conception des états

Description des états

L'état du jeu est représenté par un ensemble d'éléments fixes, la carte du jeu et les bâtiments sont déjà présents sur la carte. Un ensemble d'éléments mobiles qui sont les différents personnages peuvent se déplacer. Ces éléments ont en commun les caractéristiques suivantes.

- une coordonnées (x, y) dans la grille de la carte
- un identifiant qui permet de les différencier entre eux

Etats éléments fixes

La carte est divisée en plusieurs cases par une grille de 16x16 pixels.

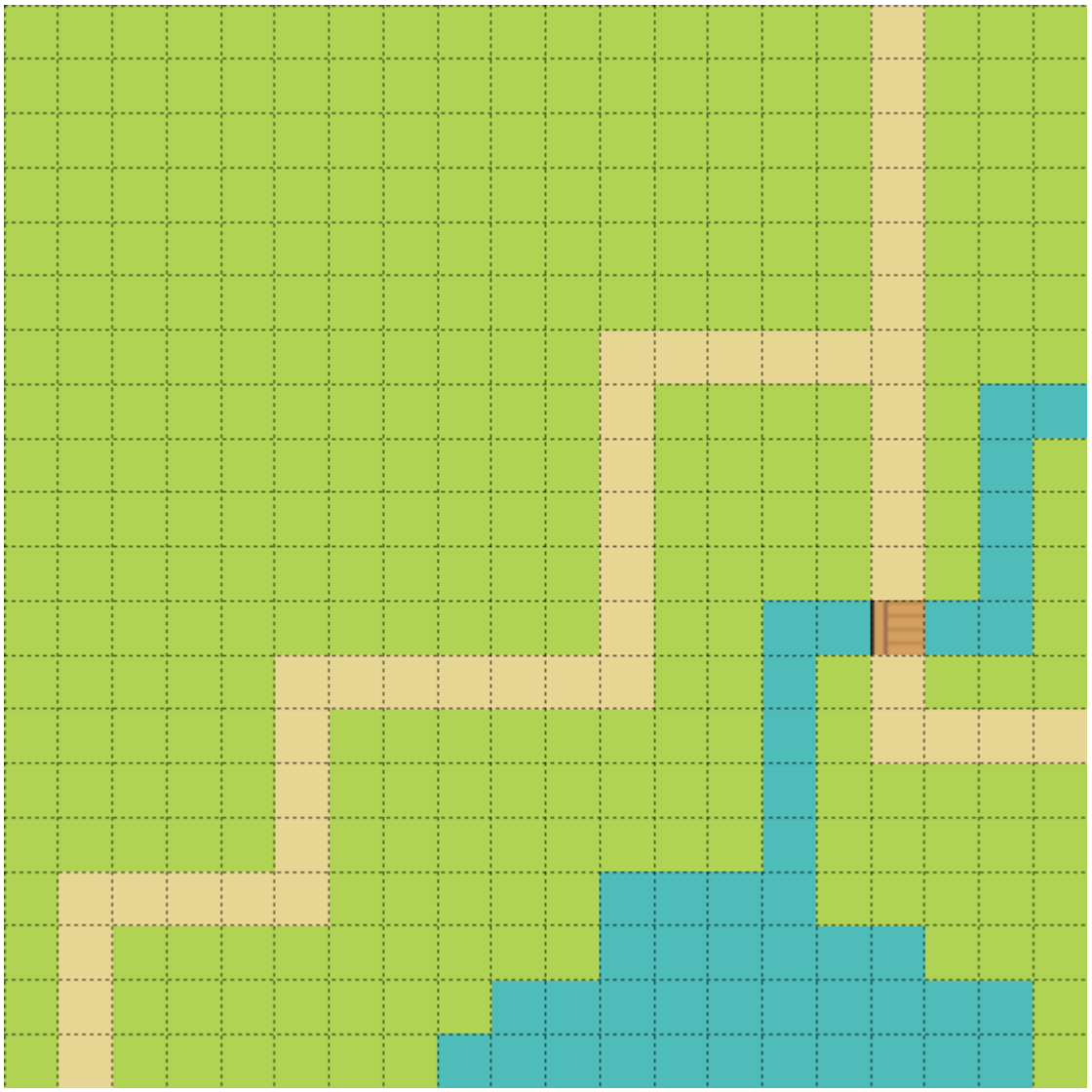

Figure 7 -La carte divisée en plusieures cases

Il existe plusieurs types de cases, et qui ont différentes caractéristiques.

- Les cases “sol” où les personnages peuvent se déplacer dessus. On y retrouve plusieurs sous catégories : côte, colline, sol, route, forêt et pont. Mis à part le côté esthétique, elles confèrent des bonus aux unités qui les traversent. Par exemple, un personnage se déplace plus vite sur la route mais moins vite en forêt
- Les cases “eau” où les personnages qui ne peuvent pas voler ne peuvent pas se déplacer dessus
- Les cases “bâtiment” qui ont des points de vie et des dégâts et où les personnages ne peuvent pas se déplacer dessus mais peuvent effectuer des actions dessus (attaquer, capturer). On distingue deux types de bâtiment : la base et le reste des bâtiments. La base est l’élément à défendre, elle possède donc plus de points de vie et de dégâts que les autres

Etats éléments mobiles

Les éléments mobiles sont les soldats qu’on peut faire apparaître à chaque tour. Ils possèdent tous les mêmes attributs, mais pas les mêmes valeurs. Les attributs sont les suivants.

- points de vie
- dégâts
- peut se déplacer
- peut attaquer
- peut capturer des bâtiments
- peut ignorer le terrain i.e. peut se déplacer sur les cases d’eau
- texture

Ils sont tous contrôlables par le joueur et peuvent uniquement se déplacer case par case. Il s’agit du joueur, de l’adversaire ou de l’IA qui décident de leur déplacement.

Etat général

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

- “turn” qui indique le nombre de tours.
- “end” qui indique la fin du jeu.

- “win” qui indique que le joueur a gagné.
- “lose” qui indique que le joueur a perdu.

ces propriétés seront encodées dans un entier pour déterminer l’état de la partie.

Conception Logicielle

Dans un premier temps nous allons concevoir la classe qui représente tout élément affichable et sélectionnable de notre jeu.

Une classe “World” qui permet d'afficher la carte. Elle contient de nombreux attributs concernant les paramètres de la carte. Elle a une relation d’agrégation avec la classe “WorldHandler”.

Attributs	Type	Explication
_Name	String	Le nom de la map
_ResPath	String	Le chemin d’accès du fichier CSV pour générer la map
_CellSize	vecteur de dimension deux (matrice)	Représente la dimension d’une cellule dans la map
_CellN	vecteur de dimensions deux (matrice)	Représente le nombre de cellules en longueur et en largeur de la map
_ SolidTexture	liste de texture	Contient les nombres Textures sur la carte

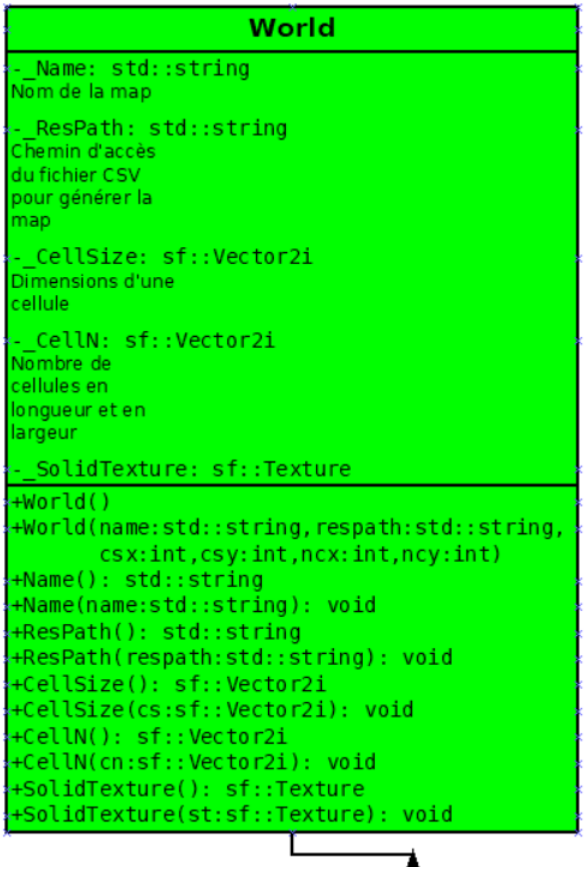

Figure 8-class “world”

Nous avons ensuite la classe “WorldHanldler” qui gère le nombre de tours, met la fin de jeux, juge quelle joueur gagne etc.

WorldHandler suit le pattern design “Traits”, met à disposition de toutes les classes la possibilité d'exécuter des fonctions lors d’événements.

Lorsque le tour commence, les fonctions contenues dans la liste TurnBeginEvents seront exécutées de même à la fin d’un tour avec TurnEndEvents.

Ainsi une classe peut s'abonner à WorldHandler en définissant les fonctions statiques OnTurnBegin et OnTurnEnd. A chaque début de tour WorldHandler éxécutera ces fonctions dans de Routine pour toutes les classes qui sont abonnées.

Le Status de la partie est encodé sur un char :

Si la partie est cours en alors Status = 0x10 + id_current_player

Si la partie est terminée alors Status = 0x20 + id_winner

Attribut	Type	Explication
CurrentWord	pointeur de type de la classe World	La map actuellement chargée
Turn	Entier(int)	Le tour de la partie
Players	Liste des classe Player	une liste des joueurs
MyID	Entier(int)	identifiant du joueur
Instance	Entier (int)	identifiant des parties en cours qui permet associer les joueurs en chaque partie
Status	Type caractère (char)	Représente le statut du jeu: si le jeu est terminé, si l'on gagne.
TurnBeginEvents	Liste de fonctions	Contient la liste des fonctions qui permettent de manipuler les événements au début des jeux
TurnBeginAsyncEvent	Liste de fonctions	Contient la liste des fonctions asynchrones qui permettent de manipuler les événements au début du jeu
TurnEndEvents	Liste de fonctions	Contient la liste des fonctions qui permettent de manipuler les événements à la fin du jeu
TurnEndAsyncEvent	Liste de fonctions	Contient la liste des fonctions asynchrones qui permet de manipuler les événements à la fin du jeu

Figure 9: la classe “WorldHandler”

La classe Player contient tous les éléments concernant les joueurs : les noms(_Name) , les identifiants (_ID). Elle a une relation d'agrégation avec classe "WorldHandler".

Attribut	Type	Fonction
_Name	String	Le nom du joueur
_ID	String	L'identifiant du joueur

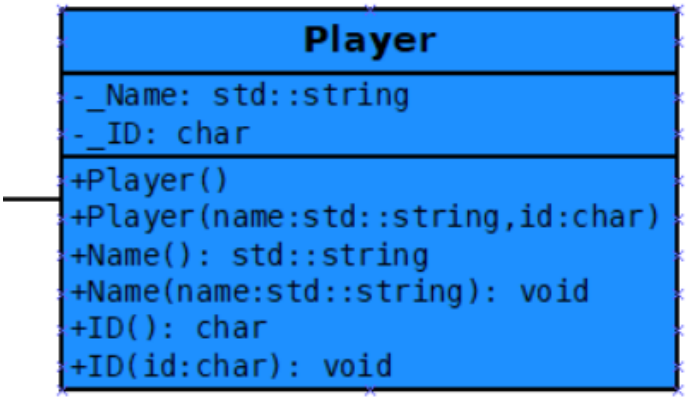

Figure 10 : La class "Player"

La classe Manager permet de gérer tous les éléments sur la carte . Il a une relation d'agrégation avec la class "Worldhandler" . Les Managers mettent à jour les éléments du jeu tout au long de la partie.

Attribut	Type	Fonction
_Name	String	Nom du manager
_ID	String	Identifiant du manager
_Elements	Liste de la classe manageable	Les listes des 'objets géré par la classe "Manager"
Managers	Vecteur de pointeur de la class "Manager"	Liste référençant tous les manager existants

Figure 10: La class “Manager”

La classe “Manageables” qui contient tous les éléments qui pourraient être manipulés par la classe “Manager”. Les Manageables sont des éléments qui peuvent être affichés et sélectionnés durant la partie. Ceci peuvent être des personnages, des éléments d’interface, les tuiles composants la map.

Attributs	Type	Explication
_Name	String	Le nom de la Manageable
_ResPath	String	Le chemin d’accès vers texture
_ID	Type entier (int)	identifiant des objets de manageable
_Render	boolean	indiquer si l’objet doit être affiché
_Selected	boolean	indiquer si l’objet est sélectionné
_Texture	vecteur des pointeurs de	Contient les pointeurs

	texture	des textures pour présenter les objets dans la map
_Sprite	liste des sprites	liste des objets associées aux textures
_Position	vecteurs de dimension deux (matrice)	représente les coordonnée des objets manipulable sur la map
_Scale	vecteurs de dimension deux (matrice)	représente la tailles des objets manageables.

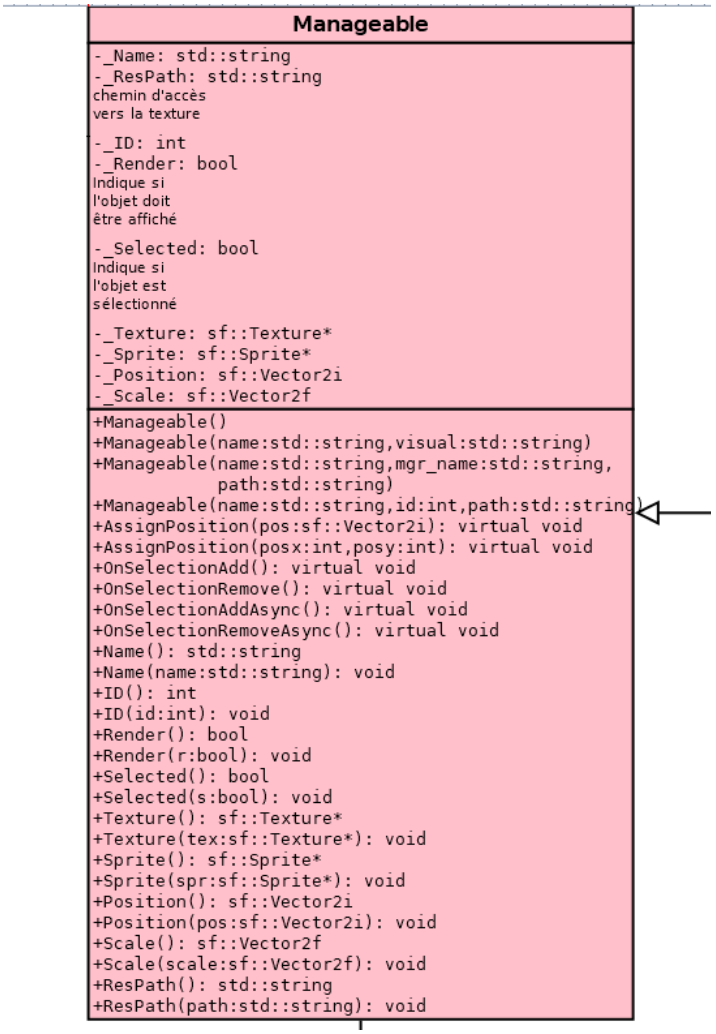

Figure 11 : La classe “Manageable”

La classe “Actor” contient toutes les informations des personnages et des bâtiments. Elle reste assez vague pour le moment car les acteurs seront chargés depuis des fichiers.

Attribut	Type	Fonction
_HP	Type entier (int)	La vie
_DMG	Type entier (int)	puissance de l'attaque
_DEF	Type entier (int)	puissance de défense
_AP	Type entier (int)	point d’action
_MP	Type entier (int)	point de movement

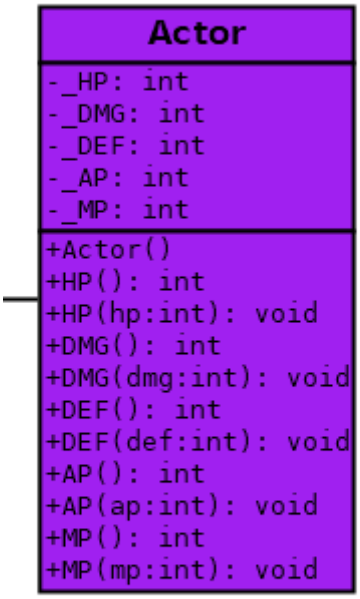

Figure 12: La class “Actor”

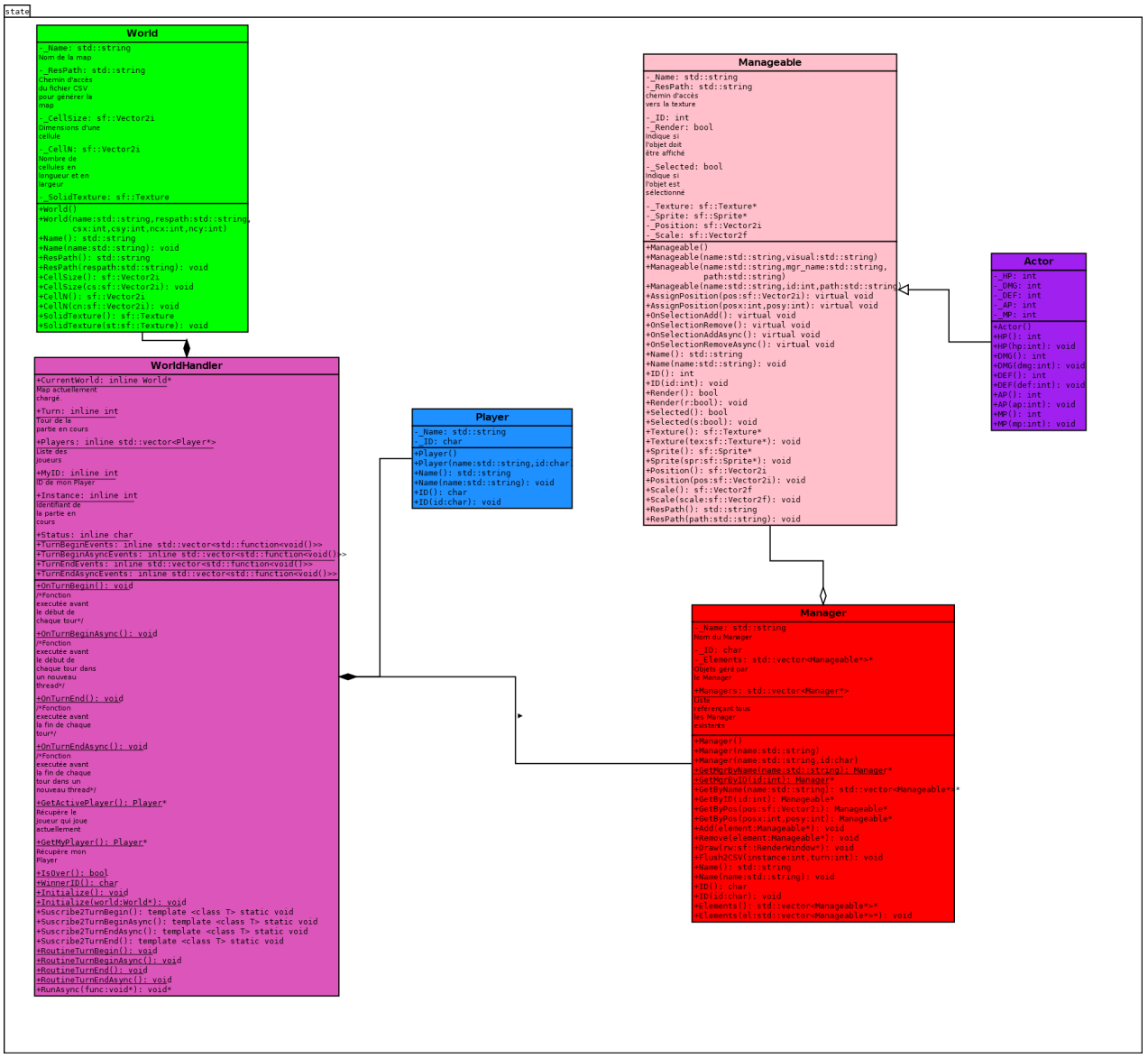

Figure 13: Le diagramme des classes.