

Projet royal war

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



Figure 1-Exemple du jeu que nous voudrions faire

| | |
|--|-----------|
| 1 Présentation Générale | 3 |
| 1.1 Archétype | 3 |
| 1.2 Règles | 3 |
| 1.3 Textures | 3 |
| 1.3.1 Terrain | 3 |
| 1.3.2 Décor | 4 |
| 1.3.3 Nature | 4 |
| 1.3.4 Projectile | 4 |
| 1.3.5 Bâtiment | 5 |
| 1.3.6 Personnages | 6 |
| 1.4 Résultat | 7 |
| 2 Description et conception des états | 8 |
| 2.1 Description des états | 8 |
| 2.1.1 Etats éléments fixes | 8 |
| 2.1.2 Etats éléments mobiles | 9 |
| 2.1.3 Etat général | 10 |
| 2.2 Conception Logicielle | 10 |
| 2.2.1 La classe "World" | 10 |
| 2.2.2 La classe "WorldHanlder" | 12 |
| 2.2.3 La classe "Player" | 15 |
| 2.2.4 La classe "Manager" | 15 |
| 2.2.5 La classe "Manageables" | 16 |
| 2.2.6 La classe "Actor" | 18 |
| 2.2.7 Le diagramme de classe pour le state | 19 |
| 3 Rendu : Stratégie et Conception | 21 |

| | |
|--|-----------|
| 3.1 Stratégie de rendu d'un état | 21 |
| 3.2 Conception logicielle | 25 |
| 3.2.1 Classe MainFrame | 25 |
| 3.2.2 Classe FileHandler | 26 |
| 4 Règle de changement d'état et moteur du jeu | 28 |
| 4.1 Changements externes | 28 |
| 4.1.1 Mise en place des acteurs et actions associées | 28 |
| 4.1.2 Script associé à chaque carte | 31 |
| 4.1.3 Commande sélection des acteurs | 32 |
| 4.1.4 Commande des actions | 33 |
| 4.2 Changement autonome | 34 |
| 4.3 Conception logicielle | 34 |

1 Présentation Générale

1.1 Archétype

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

1.2 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

1.3 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`

1.3.1 Terrain



Figure 2- Texture pour les terrains

1.3.2 Décor

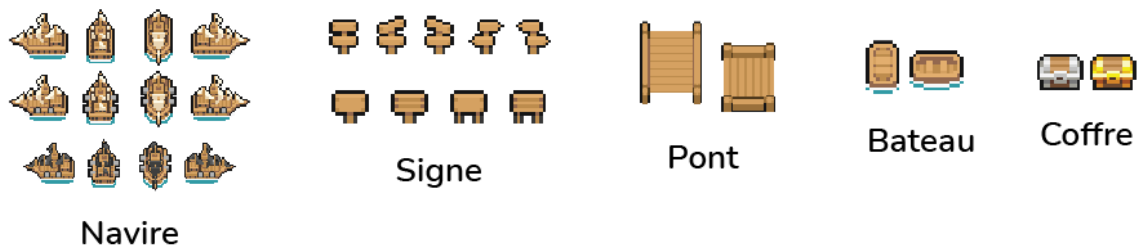


Figure 3 -Texture pour les décors

1.3.3 Nature



Figure 4 -Texture pour la nature

1.3.4 Projectile



Figure 5 -Texture pour la projectile

1.3.5 Bâtiment

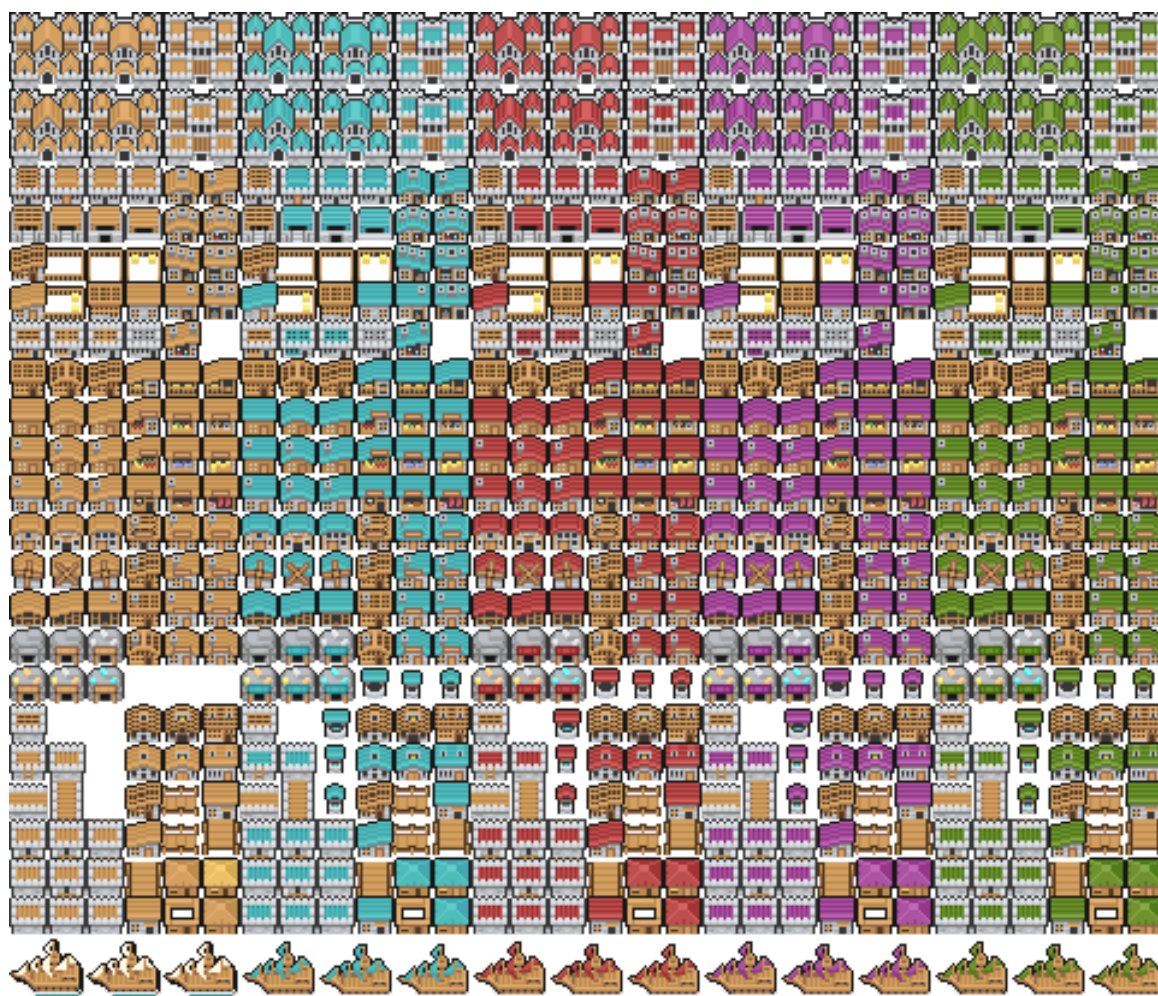


Figure 6 -Texture pour les bâtiments

1.3.6 Personnages

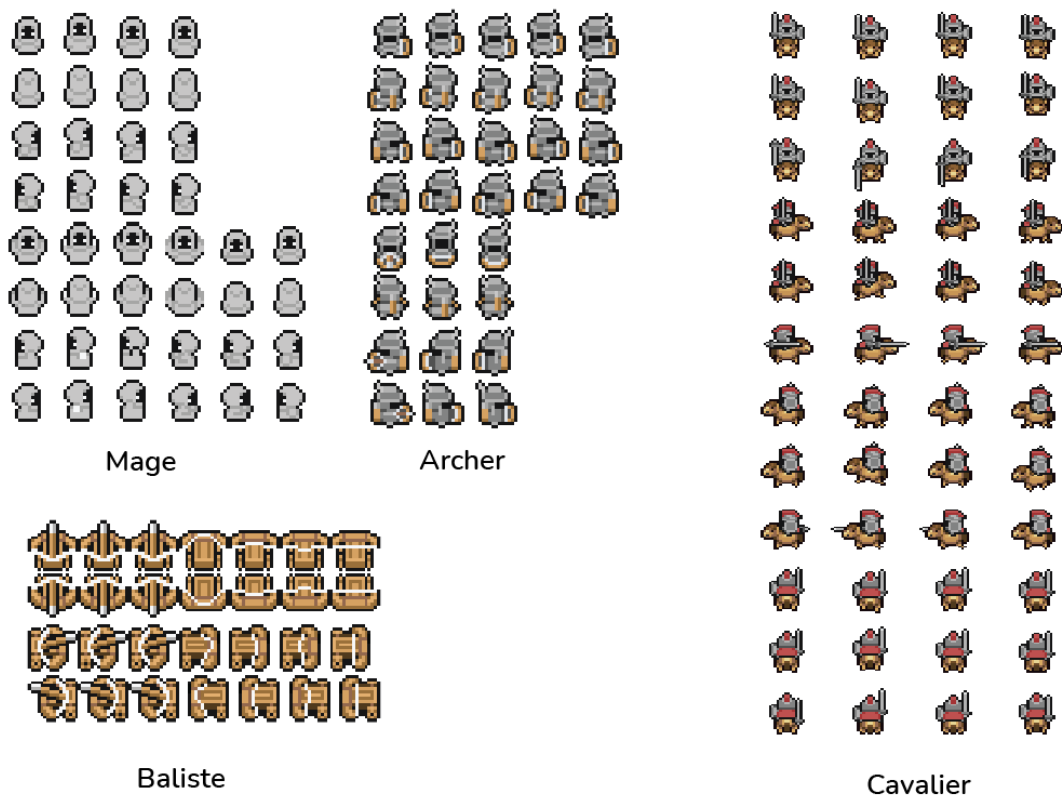


Figure 7.1 -Texture pour les personnage

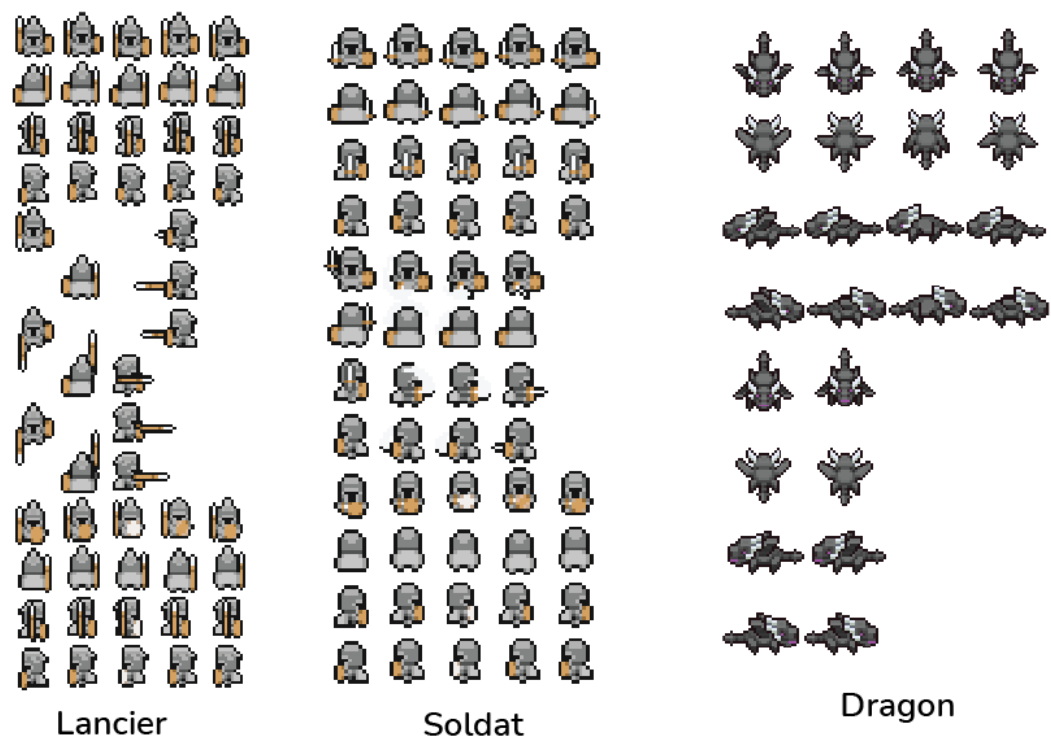


Figure 7.2 -Texture pour les personnages

1.4 Résultat

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



Figure 8 -l'exemple de la carte créée

2 Description et conception des états

2.1 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

2.1.1 Etats éléments fixes

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

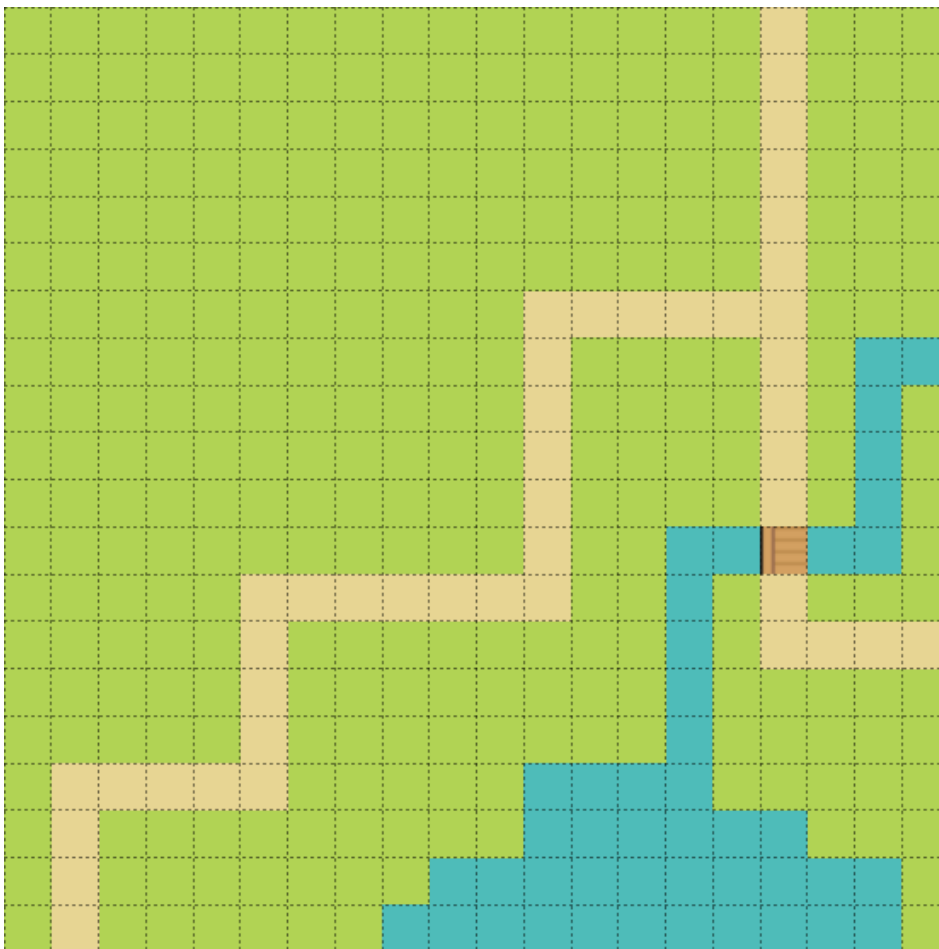


Figure 9 -La carte divisée en plusieurs cases

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

2.1.2 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(points d'attaque)
- 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.

2.1.3 Etat général

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

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.

2.2 Conception Logicielle

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

2.2.1 La classe “World”

Une classe “World” qui permet d'afficher la carte. Elle contient de nombreux attributs concernant les paramètres de la carte, décrits dans le tableau suivant.. Elle a une relation d'agrégation avec la classe “WorldHandler”.

Tab 1-Le tableau des attributs contenus dans la classe “World”

| Attributs | Type | Explication |
|---------------|--------------------------------------|--|
| _Name | String | Nom de la map |
| _ResPath | String | 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 |

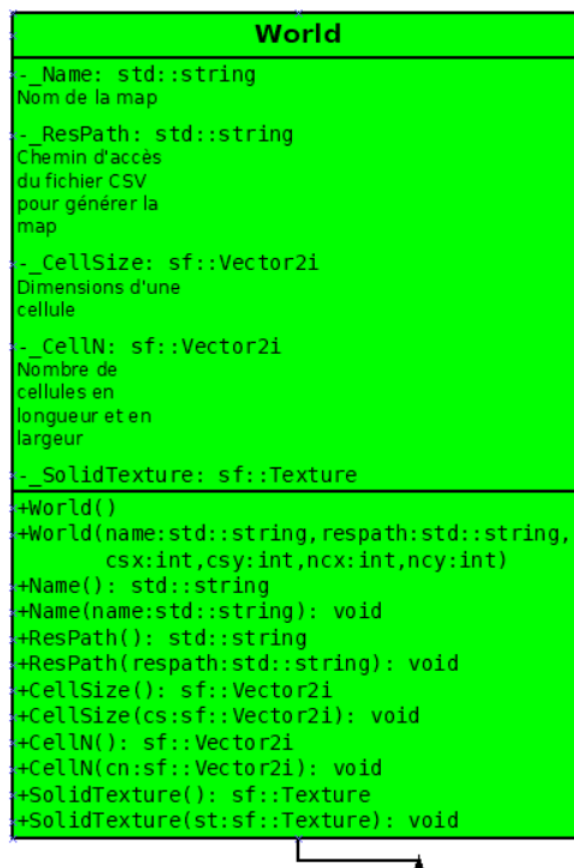


Figure 10-class “World”

2.2.2 La classe “WorldHandler”

WorldHandler suit le pattern désigne “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 exécutera 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 $\text{Status} = 0x10 + \text{id_current_player}$

Si la partie est terminée alors $\text{Status} = 0x20 + \text{id_winner}$

Tab 2 -Le tableau des attributs contenus dans la classe “WorldHandler”

| Attribut | Type | Explication |
|---------------------|-------------------------------------|--|
| CurrentWorld | Pointeur de type de la classe World | Map actuellement chargée |
| Turn | Entier(int) | Tour de la partie |
| Players | Liste des classe Player | 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 |

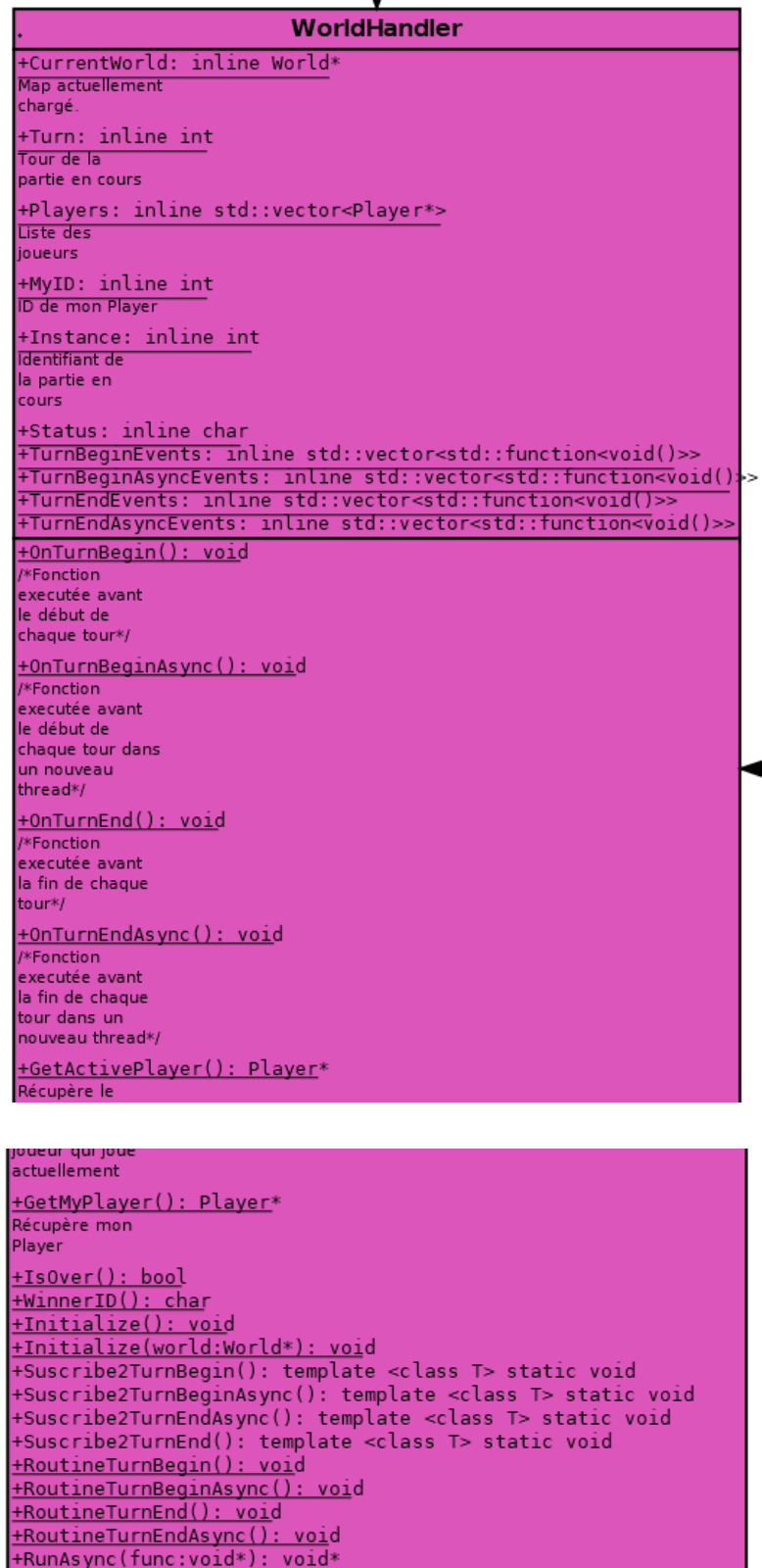


Figure 11: la classe “WorldHandler”

2.2.3 La classe “Player”

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”.

Tab 3-Le tableau des attributs dans la classe “Player”

| Attribut | Type | Fonction |
|----------|--------|-----------------------|
| _Name | String | Nom du joueur |
| _ID | String | Identifiant du joueur |

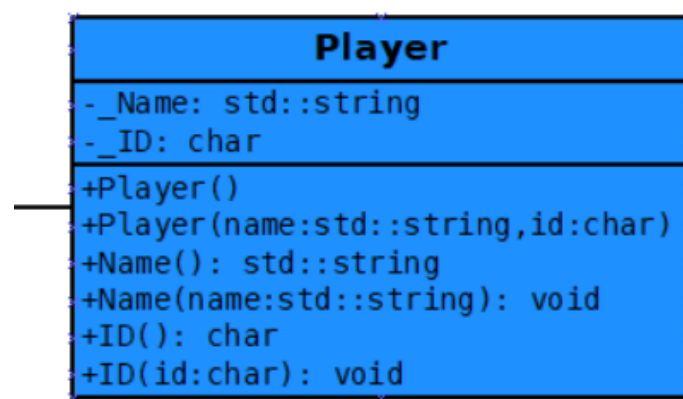


Figure 12 : La class “Player”

2.2.4 La classe “Manager”

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

Tab 4- Le tableau des attributs contenus dans la classe “Manager”

| Attribut | Type | Fonction |
|-----------|---|---|
| _Name | String | Nom du manager |
| _ID | String | Identifiant du manager |
| _Elements | Liste de la classe manageable | Listes des 'objets géré par la classe “Manager” |
| Managers | Vecteur de pointeur de la class “Manager” | Liste référençant tous les manager existants |

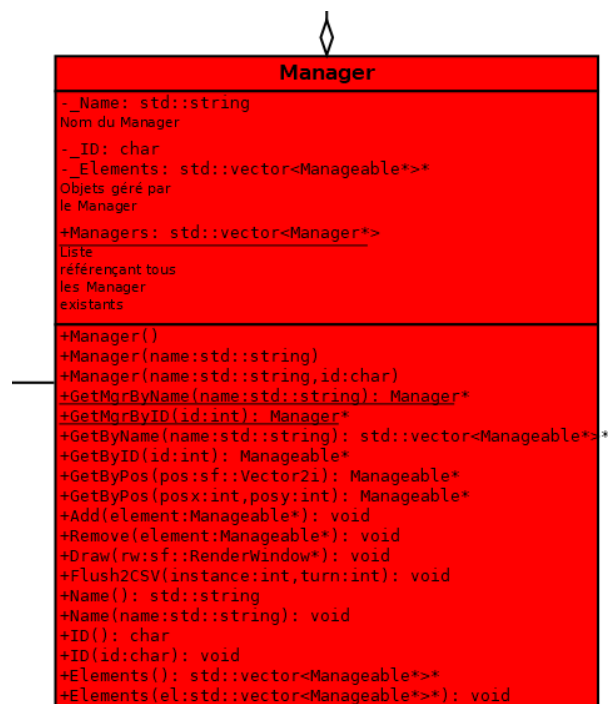


Figure 13: La class “Manager”

2.2.5 La classe “Manageables”

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.

Tab 5-Le tableau des attributs contenus dans la classe “Manageable”

| Attributs | Type | Explication |
|-----------|--------------------------------------|---|
| _Name | String | Nom de Manageable |
| _ResPath | String | Chemin d'accès vers texture |
| _ID | Type entier (int) | Identifiant des objets de manageable |
| _Render | Boolean | Indique si l'objet doit être affiché |
| _Selected | Boolean | Indique si l'objet est sélectionné |
| _Texture | Vecteur des pointeurs de texture | Contient les pointeurs 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 coordonnées des objets manipulable sur la map |
| _Scale | Vecteurs de dimension deux (matrice) | Représente la taille des objets manageables. |

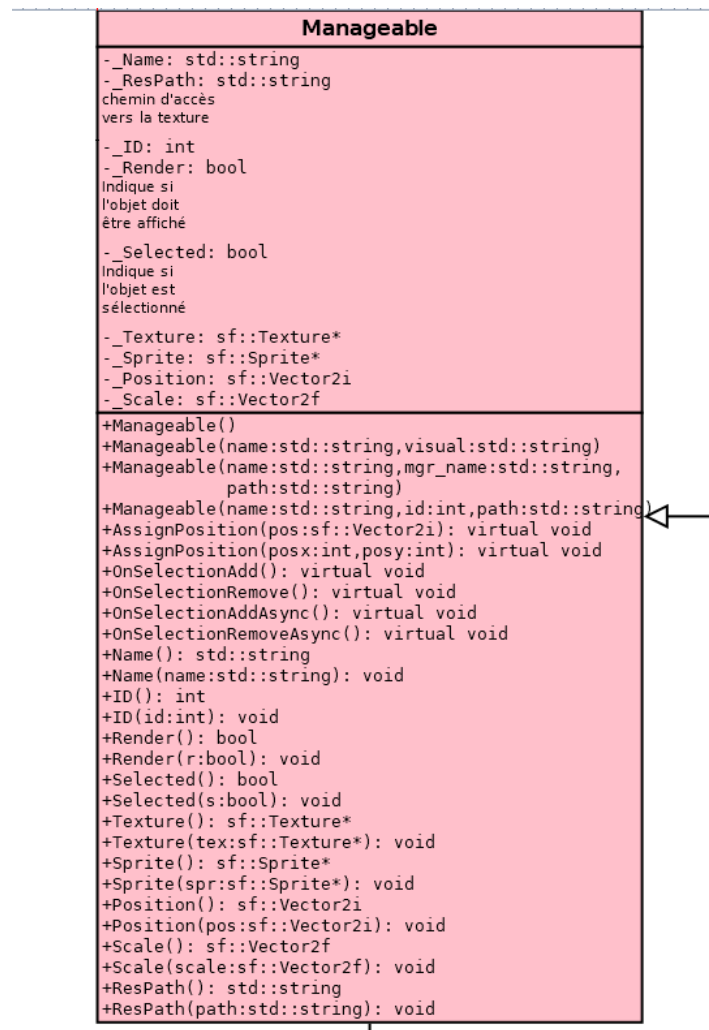


Figure 14 : La classe “Manageable”

2.2.6 La classe “Actor”

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.

Tab 6-Le tableau des attributs contenus dans la classe “Actor”

| Attribut | Type | Fonction |
|----------|-------------------|------------------------|
| _HP | Type entier (int) | Point de 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 mouvement |

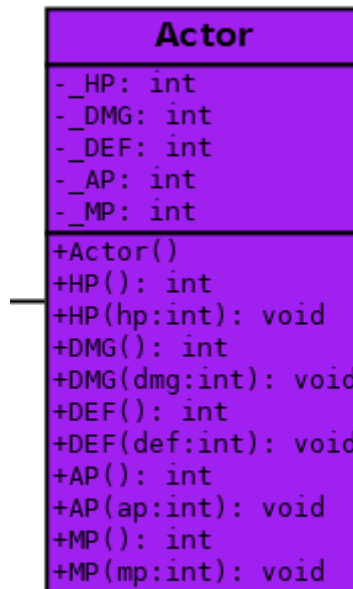


Figure 15: La class “Actor”

2.2.7 Le diagramme de classe pour le state

Nous rassemblons toutes les classes citées ci-dessus pour obtenir le diagramme de classe pour l' état state comme la figure 16.

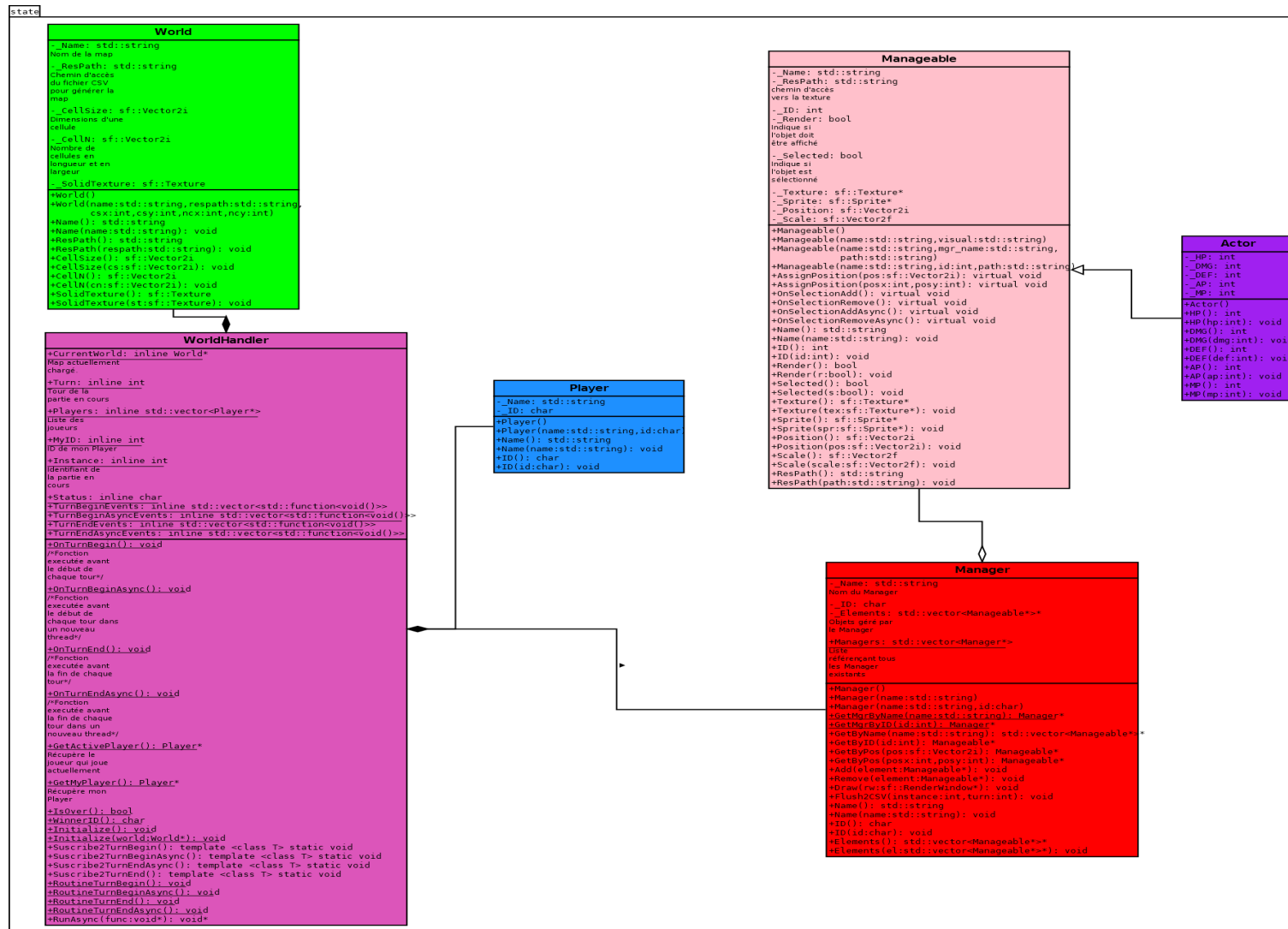


Figure 16: Le diagramme des class

3 Rendu : Stratégie et Conception

3.1 Stratégie de rendu d'un état

Un état de notre jeu comprend au minimum 3 éléments : la map, les personnages et les informations du jeu (nombre de tours, ...). Selon la situation on peut avoir un autre élément qui est une fenêtre d'actions réalisables par les personnages.

Pour le rendu des états nous utiliserons la bibliothèque SFML ainsi qu'un rendu à l'aide de tuiles. Notre affichage se fait en utilisant plusieurs couches superposées les unes aux autres:

- La couche 0 est une couche qui charge toutes les ressources du jeu. Pour affecter une texture, il suffit d'utiliser un pointeur vers la texture.
- La 1ère couche est la map du jeu
- La 2ème couche est celle où se trouve les personnages ainsi que les bâtiments
- La 3ème couche servira à afficher le menu des actions
- La dernière couche sera celle où sera affiché les informations du jeu

Utiliser plusieurs couches va nous permettre de superposer les éléments et de faire des sélections traversantes nécessaires pour la mise en surbrillance, l'affichage d'un effet visuel ou l'affichage d'un personnage sur une tile. Le nombre de couches est déterminé dans le fichier *Managers.csv*, on peut en ajouter autant qu'on veut.

Pour la construction de la map nous associons un id à un certain sprite afin de construire une tuile. Nous mettons ces id ensuite dans un fichier CSV. Le changement de la map se fait donc en changeant l'id de la tuile dans le fichier CSV.

Les images des tuiles sont mises dans le dossier *res/Texture*. Nous construisons dans le dossier *src/client/table* un fichier *<<ManageablesVisuals.csv >>* où est contenu chaque identité de chaque tuile constituant la map comme dans la figure 17. Chaque identité associe un attribut *MV_ID* (une chiffre) qui permet de construire la map.

```
src > client > tables > ManageablesVisuals.csv
1  #MV_NAME,MGR_NAME,MV_ID,PATH,SCALE_X,SCALE_Y
2  BG_TILE_GRASS,ASSET_MGR,0,res/texture/grass.png,0.5,0.5
3  BG_TILE_SAND,ASSET_MGR,3,res/texture/road.png,0.5,0.5
4  BG_TILE_WATER,ASSET_MGR,1,res/texture/water.png,0.5,0.5
5  BG_TILE_BRIDGE,ASSET_MGR,2,res/texture/bridge.png,0.5,0.5
6  ACTOR_KNIGHT,ASSET_MGR,4,res/texture/knight.png,0.036,0.036
7  ACTOR_MAUSOLEUM,ASSET_MGR,5,res/texture/mausoleum.png,0.5,0.5
8  ACTOR_MAUSOLEUM2,ASSET_MGR,6,res/texture/mausoleum2.png,0.5,0.5
9  ACTOR_CYANKEEP,ASSET_MGR,9,res/texture/CyanKeep.png,0.5,0.5
10 ACTOR_REDKEEP,ASSET_MGR,10,res/texture/RedKeep.png,0.5,0.5
11 BG_TILE_STONE,ASSET_MGR,7,res/texture/stone.png,0.5,0.5
12 BG_TILE_GRASS2,ASSET_MGR,8,res/texture/grass2.png,0.5,0.5
13 BG_TILE_CLIFF11,ASSET_MGR,11,res/texture/cliff11.png,0.5,0.5
```

Figure 17 : ManageablesVisuals.csv

Tab7- Attributs des éléments dans le fichier ManageablesVisuals.csv

| Attribut | Fonction | Exemple |
|----------|---|-----------------------|
| MV_NAME | Nom de la map | BG_TILE_GRASS |
| MG_NAME | Nom qui permet à la classe Manager de manipuler les tuiles | ASSET_MGR |
| MV_ID | Chiffre permet de distinguer les images tuiles et donc de construire la map | 0 |
| PATH | Chemin pour accéder les images des tuiles | res/texture/grass.png |
| SCALE_X | Coefficient en X qui permet de convertir la taille de l'image | 0.5 |
| SCALE_Y | Coefficient en Y qui permet de convertir la taille de l'image | 0.5 |

Nous construisons un fichier csv dans le dossier *src/client/map* qui contient les "MV_ID" de chaque tuile pour créer la map. Nous avons un exemple dans la figure 18.

| | |
|----|---|
| 1 | 0,0,0,0,0,0,0,0,0,0,0,0,23,1,21,0,7,0,0,0,0,0,0,0,0,0,0,0,0,0, |
| 2 | 0,0,0,0,0,0,0,0,0,0,0,0,23,1,21,0,0,36,32,37,0,0,0,0,0,0,0,36,32,32,32, |
| 3 | 0,7,0,42,42,42,42,42,42,42,0,23,1,21,0,0,23,1,2,42,42,42,42,0,0,0,23,1,1,1, |
| 4 | 0,0,0,42,0,0,44,0,0,42,0,23,1,21,0,0,35,12,34,0,0,0,42,0,42,42,2,45,1,1, |
| 5 | 0,0,0,42,36,32,37,0,0,42,0,23,1,21,0,0,44,0,0,0,0,0,0,42,0,23,1,11,12, |
| 6 | 0,0,0,42,2,1,21,0,0,42,0,23,1,21,0,0,0,0,42,42,42,42,42,0,42,0,23,1,21,0, |
| 7 | 0,0,0,7,35,12,34,0,0,42,0,23,1,21,0,0,0,0,42,0,0,0,42,0,42,0,23,1,21,0, |
| 8 | 0,0,0,0,0,0,44,0,0,42,42,151,152,153,42,42,42,42,0,0,0,42,42,42,151,152,153,42, |
| 9 | 0,0,0,0,0,0,0,0,0,0,42,42,161,162,163,42,42,0,0,0,0,0,0,0,0,0,0,161,162,163,42, |
| 10 | 0,0,42,42,42,42,0,0,0,42,0,23,1,21,0,42,0,0,0,7,0,0,0,0,0,0,23,1,21,0, |
| 11 | 0,0,42,0,0,42,0,0,0,42,0,23,1,21,0,42,0,0,0,0,0,0,0,0,0,0,23,1,21,0, |
| 12 | 0,0,42,0,0,42,42,42,42,42,0,23,1,21,0,42,0,44,0,0,0,7,0,0,0,0,23,1,21,0, |
| 13 | 0,0,42,0,0,0,0,0,0,42,0,23,1,21,0,42,42,42,42,42,42,42,0,0,0,23,1,21,0, |
| 14 | 0,0,42,0,0,0,0,0,0,0,0,23,1,21,0,0,42,0,0,0,0,0,42,0,0,0,23,1,21,0, |
| 15 | 0,0,42,42,0,36,32,32,111,112,32,33,1,31,32,32,111,112,32,32,32,32,111,112,32,32,33,1,31,32, |
| 16 | 0,0,0,42,42,43,1,1,121,122,1,1,1,1,1,1,121,122,1,1,1,1,121,122,1,1,1,1,1,1, |
| 17 | 0,44,0,42,42,43,1,11,131,132,12,12,12,12,12,12,131,132,12,12,12,12,131,132,12,12,12,12,12,12, |
| 18 | 0,0,0,42,42,43,45,21,0, |
| 19 | 0,0,0,42,42,43,45,21,0,0,0,0,0,0,7,0,0,0,0,44,0,0,0,0,0,0,0,0,0,0,0, |

Figure 18 : Fichier WorldTest.csv qui permet créer la map.

Nous définissons les paramètres des maps dans le fichier *Worlds.csv* dans le dossier *src/client/tables* comme dans la figure 19:

```
1 #WORLD_NAME,MAP_PATH,CELL_X,CELL_Y,N_CELL_X,N_CELL_Y
2 STD_WORLD,src/client/maps/DefaultWorld.csv,41,41,19,19
3 O_WORLD,src/client/maps/OtherWorld.csv,41,41,19,19
4 EVERGREEN,src/client/maps/Evergreen.csv,41,41,10,10
5 POOL,src/client/maps/Pool.csv,41,41,11,10
6 WORLD_TEST,src/client/maps/WorldTest.csv,41,41,30,19,
7
```

Figure 19: Fichier Worlds.csv

Tab 8-Attributs des éléments dans le fichier Worlds.csv

| Attribut | Fonction | Exemple |
|------------|---|------------------------------|
| WORLD_NAME | Nom de map qui sert à changer la map pour l'affichage | STD_WORLD |
| MAP_PATH | Chemin d'accès pour trouver la map | src/client/maps/DefaultWorld |

| | | |
|----------|---|----|
| CELL_X | Dimension sur l'axe X d'une tile (width) | 41 |
| CELL_Y | Dimension sur l'axe Y d'une tile (height) | 41 |
| N_CELL_X | Nombre de tuile affichées sur l'abscisse axe X | 19 |
| N_CELL_Y | Nombre de tuiles affichées sur l'ordonnée axe Y | 19 |

Afin d'afficher la map, nous devons changer la première ligne du fichier *LaunchArgs.csv* situé dans le dossier *src/client/tables* comme dans la figure 20. Nous mettons à chaque fois le nom de la map après *SCENE* pour afficher la map.


```
src > client > tables >  LaunchArgs.csv
1  SCENE,WORLD_TEST
2  IP_SERVER,127.0.0.1
3  IP_PORT,XXXX
4  FLUSH_PATH,src/client/BoardSnapshot
```

Figure 20: Fichier LaunchArgs.csv

3.2 Conception logicielle

Nous utilisons principalement deux classes pour le rendu d'un état.

3.2.1 Classe MainFrame

La classe *MainFrame* va créer la fenêtre qui contiendra notre rendu.

Tab 9-Attributs de la classe MainFrame

| Attribut | Type | Fonction |
|-----------|--------|------------------------|
| Name | String | Nom de la fenêtre |
| Path | String | Chemin |
| Width | Int | Largeur de la fenêtre |
| Height | Int | Longueur de la fenêtre |
| FrameRate | Int | FPS de la fenêtre |

```
MainFrame  
Fenetre du jeu  
- _Name: std::string  
- _Height: int  
- _Width: int  
- _Framerate: int  
- _Window: sf::RenderWindow*  
+MainFrame(name:std::string,height:int,width:int)  
+~MainFrame()  
+Tick(): void  
fonction  
exécutée à  
chaque frame.  
+Draw(): void  
Dessine les  
objets contenus  
dans les  
Managers  
+Start(): void  
Fonction  
exécutée au  
lancement de la  
fenetre  
+Name(n:std::string): void  
+Name(): std::string  
+Height(): int  
+Height(h:int): void  
+Width(): int  
+Width(w:int): void  
+Framerate(): int  
+Framerate(f:int): void  
+InitWorld(): void  
+InitActors(): void  
+Window(win:sf::RenderWindow*): void  
+Window(): sf::RenderWindow*
```

Figure 21: La classe MainFrame

Tick() est une fonction exécutée à chaque frame.

Draw() est une fonction qui permet de dessiner les objets contenus dans les managers.

Start() est une fonction exécutée une fois au lancement de la fenêtre.

InitWorld() est une fonction qui permet d'initialiser la carte.

InitActors() est une fonction qui permet d'initialiser les acteurs.

Les restes des fonctions sont des "getter" et "setter" pour les attributs.

3.2.2 Classe FileHandler

La classe FileHandler est celle qui va lire nos fichiers CSV et ainsi créer nos éléments pour pouvoir les afficher.

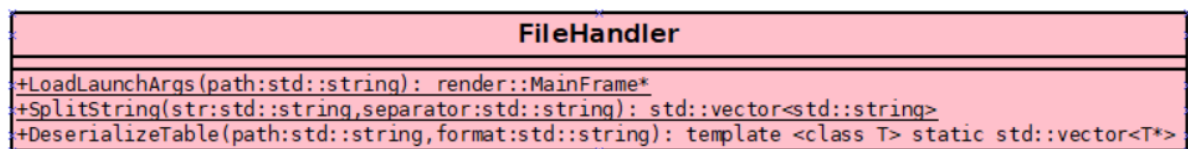


Figure 22: Classe FileHandler.

SplitsString() est une fonction qui permet de séparer les chaînes.

LoadLaunchArgs() est une fonction qui lance une fenêtre selon les paramètres du fichier de configuration *LaunchArgs.csv*.

DeserializeTable() est une fonction pour désérialiser les lignes d'un fichier dans une liste d'objet de la classe spécifiée en paramètre de template (exemple `DeserializeTable<Actor>("Actors.csv","CSV")`). Cette fonction nous permet de charger les tables aisément.

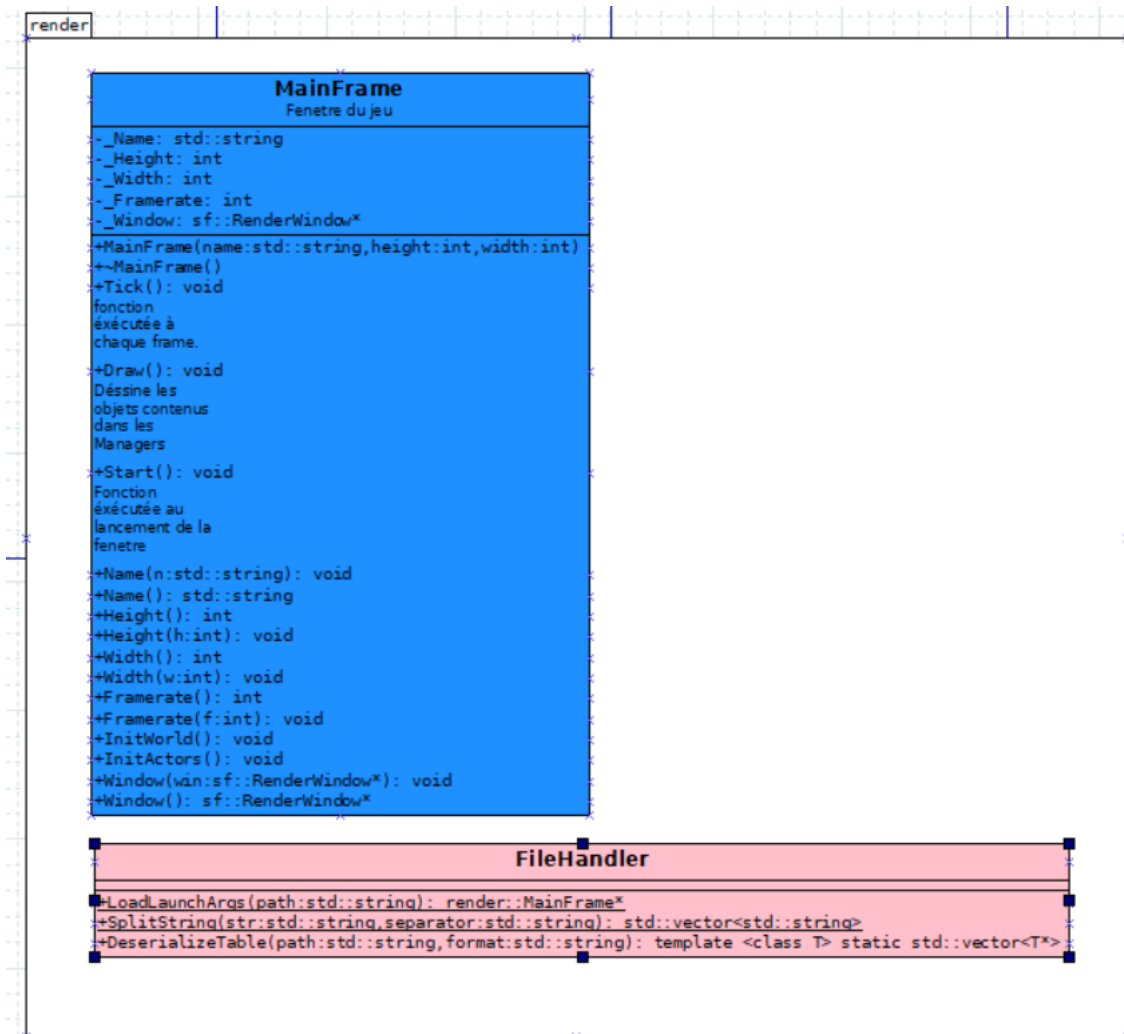


Figure 23: Diagramme du namespace rende

4 Règles de changement d'état et moteur du jeu

4.1 Changements externes

Les changements extérieurs sont provoqués par des commandes, en particulier la pression d'une touche de clavier ou le clic d'une souris. Nous distinguons 2 types de commandes extérieures :

- Commande de sélection des acteurs.
- Commande des actions

4.1.1 Mise en place des acteurs et actions associées

Tous les personnages sont définis sous le fichier *Actors.csv* dans le dossier *src/client/tables*, comme montré ci-dessous.

```
1  #ACTOR_NAME,VISUALS,ACTOR_HP,ACTOR_DMG,ACTOR_DEF,ACTOR_ACT_PTS,ACTOR_MOV_P
2  BUILDING_MAUSOLEUM,ACTOR_MAUSOLEUM,100,30,60,2,0,STD_ATTACK;STD_MOVE
3  BUILDING_MAUSOLEUM2,ACTOR_MAUSOLEUM2,100,30,60,2,0,STD_ATTACK;STD_MOVE
4  BUILDING_CYANRESSOURCE,ACTOR_CYANRESSOURCE,100,40,50,2,0,STD_ATTACK;STD_MO
5  BUILDING_REDRESSOURCE,ACTOR_REDRESSOURCE,100,40,50,2,0,STD_ATTACK;STD_MOVE
6  BUILDING_CYANKEEP,ACTOR_CYANKEEP,100,20,70,2,0,STD_ATTACK;STD_MOVE
7  BUILDING_REDKEEP,ACTOR_REDKEEP,100,20,70,2,0,STD_ATTACK;STD_MOVE
8  HERO_CYANBOWMAN,ACTOR_CYANBOWMAN,100,55,25,3,3,STD_ATTACK;STD_MOVE
9  HERO_REDBOWMAN,ACTOR_REDBOWMAN,100,55,25,3,4,STD_ATTACK;STD_MOVE
10 HERO_CYANMAGE,ACTOR_CYANMAGE,100,70,35,5,3,STD_ATTACK;STD_MOVE
11 HERO_REDMAGE,ACTOR_REDMAGE,100,70,35,5,3,STD_ATTACK;STD_MOVE
12 HERO_REDSPEARMAN,ACTOR_REDSPEARMAN,100,80,35,5,6,STD_ATTACK;STD_MOVE
13 HERO_CYANSPEARMAN,ACTOR_CYANSPEARMAN,100,80,35,5,6,STD_ATTACK;STD_MOVE
14 HERO_CYANKNIGHT,ACTOR_CYANKNIGHT,150,80,45,6,8,STD_ATTACK;STD_MOVE
15 HERO_REDKNIGHT,ACTOR_REDKNIGHT,150,80,45,6,8,STD_ATTACK;STD_MOVE
16 HERO_CYANDRAGON,ACTOR_CYANDRAGON,200,90,65,5,10,STD_ATTACK;STD_MOVE
17 HERO_REDDRAGON,ACTOR_REDDRAGON,200,90,65,5,10,STD_ATTACK;STD_MOVE
```

Figure 24 : Fichier "Actors.csv"

Tab 10-Attributs des éléments du fichier Actors.csv

| Attribut | Type | Fonction |
|---------------|--------|--|
| ACTOR_NAME | String | Nom de l'acteur |
| VISUALS | String | Nom de sa texture qui a été déclaré au préalable dans le fichier ManageablesVisuals.csv |
| ACTOR_HP | Int | Point de vie de l'acteur |
| ACTOR_DMG | Int | Point d'attaque de l'acteur |
| ACTOR_DEF | Int | Point de défense de l'acteur |
| ACTOR_ACT_PTS | Int | Point d'action de l'acteur |
| ACTOR_MOV_PTS | Int | Point de mouvement de l'acteur |
| ACTIONS | String | Scripts des actions à associer avec cet acteur, par exemple un script pour attaquer, se défendre ou spawn des personnages Chaque script différent doit être séparé par une virgule |

Concernant les scripts des actions, ils sont déclarés dans le fichier *Actions.csv*, comme ci-dessous.

```
src > client > tables > Actions.csv
1  #ACTION_NAME,OPCODE,ACTION_POINTS,MOVEMENT_POINTS,PATTERN,NET_CMD
2  STD_ATTACK,0x1000,1,0,PATTERN_DIAMOND4,NET_CMD_ATTACK
3  STD_MOVE,0x2000,0,1,PATTERN_DIAMOND2,NET_CMD_MOVE
4  STD_INVOKE_CYAN,0x3000,1,0,PATTERN_CROSS,NET_CMD_INVOKE_CYAN
5  STD_INVOKE_RED,0x3001,1,0,PATTERN_CROSS,NET_CMD_INVOKE_RED
```

Figure 25 : Fichier "Actions.csv"

Tab 11-Attributs des éléments du fichier Actions.csv

| Attribut | Type | Fonction |
|-----------------|--------|--|
| ACTION_NAME | String | Nom de l'action |
| OPCODE | String | Id de la l'action |
| ACTION_POINTS | Int | Point d'action que l'action va coûter |
| MOVEMENT_POINTS | Int | Point de mouvement que l'action va consommer |
| PATTERN | Int | Pattern associé à l'action |
| NET_CMD | Int | Nom de la commande à gérer par le moteur |

L'attribut OPCODE est l'ID de l'action. Le bit de poids fort sert à identifier le type d'action : on distingue trois principaux types, se déplacer, attaquer et invoquer. Le bit de poids faible va distinguer la sous catégorie de l'action, comme par exemple l'action invoquer pour deux acteurs différents.

L'attribut NET_COMMANDE sert à envoyer au moteur le nom de la fonction à traiter.

Chaque joueur va avoir au début de chaque tour un nombre de points d'actions limités, afin de ralentir la partie et de bien choisir les actions à effectuer. L'attribut MOVEMENT_POINTS va uniquement servir pour l'action de déplacement, puisqu'il indique jusqu'à où il est possible de se déplacer.

L'attribut PATTERN est référencé dans la fichier *Pattern.csv*, où on va déclarer les pattern des mouvements, comme ci-dessous.


```
src > client > tables >  Patterns.csv
1  #PATTERN_NAME,PATTERN_PATH,SIZE_X,SIZE_Y
2  PATTERN_CROSS,src/client/actions_patterns/PatternCrossSimple.csv,3,3
3  PATTERN_DIAMOND4,src/client/actions_patterns/PatternDiamond4.csv,9,9
4  PATTERN_DIAMOND2,src/client/actions_patterns/PatternDiamond2.csv,5,5
```

Figure 26 : Fichier "Pattern.csv"

Tous les différents pattern se trouvent dans le dossier *src/client/actions_patterns/*. Par exemple, prenons le premier pattern. Le fichier csv associé est le suivant.


```
src > client > actions_patterns >  PatternCrossSimple.csv
1    0,1,0
2    1,A,1
3    0,1,0
```

Figure 27 : Fichier "Pattern.csv"

On va déclarer la taille du motif, qui est du 3x3, et dessiner littéralement le motif. Les 1 correspondent aux cases associées au motif, et les 0 ceux qui ne le sont pas.

Ce système reste flexible, ce qui nous permet de déclarer autant d'actions avec des patterns associés que l'on souhaite, sans pour autant devoir redéfinir le dia à chaque modification.

4.1.2 Script associé à chaque carte

Nous définissons ensuite la position des personnages et les fonctions associées dans différents fichiers scripts. Par exemple, nous avons créé un fichier script *Layer_Actor.bhv* pour positionner les personnages. Nous définissons ensuite où se trouve ce script dans le fichier *Scripts.csv* du dossier *src/client/tables* comme montré ci-dessous.


```
src > client > tables >  Scripts.csv
1    #SCRIPT_NAME,PATH
2    #IS_BHV,src/client/scripts/InputSourceKeyboard.bhv
3    STD_WORLD,src/client/scripts/STD_WORLD.bhv
4    STD_SCENARIO,src/client/scripts/StandardScenario.bhv
5    LAYER_ACTOR,src/client/scripts/Layer_Actor.bhv
6    REGLE,src/client/scripts/Regle.bhv
7    SCRIPT_ACTIONS,src/client/scripts/Script_Actions.bhv
8    HOME,src/client/scripts/Home.bhv
9
```

Figure 28 : Fichier Scripts.csv

Nous associons ensuite un script à la map où nous voulons ajouter les personnages. Pour cela, nous ajoutons le nom du script déclaré dans le fichier *Scripts.csv* dans la définition des maps dans le fichier *Worlds.csv* comme ci-dessous.

```
Worlds.csv X
src > client > tables > Worlds.csv
1 #WORLD_NAME,MAP_PATH,CELL_X,CELL_Y,N_CELL_X,N_CELL_Y,SCRIPT
2 STD_WORLD,src/client/maps/DefaultWorld.csv,40,40,19,19,STD_WORLD
3 O_WORLD,src/client/maps/OtherWorld.csv,40,40,19,19,0
4 EVERGREEN,src/client/maps/Evergreen.csv,40,40,10,10,0
5 POOL,src/client/maps/Pool.csv,40,40,11,10,0
6 WORLD_TEST,src/client/maps/WorldTest.csv,40,40,30,19,LAYER_ACTOR
7 BEACH,src/client/maps/Beach.csv,40,40,26,12,LAYER_ACTOR
8 HOME,src/client/maps/Home.csv,256,64,5,9,0
```

Figure 29 : Fichier “Worlds.csv”.

Par exemple, nous associons à la map BEACH le script LAYER_ACTOR.

4.1.3 Commande sélection des acteurs

Au début de chaque tour, tous les acteurs, que ce soit un personnage ou un bâtiment, possèdent un statut disponible. Le joueur sélectionne un personnage avec le curseur que nous pouvons déplacer avec la souris. Une fois un acteur sélectionné, il est ajouté comme [caster].

Par exemple, le dragon bleu est sélectionné.

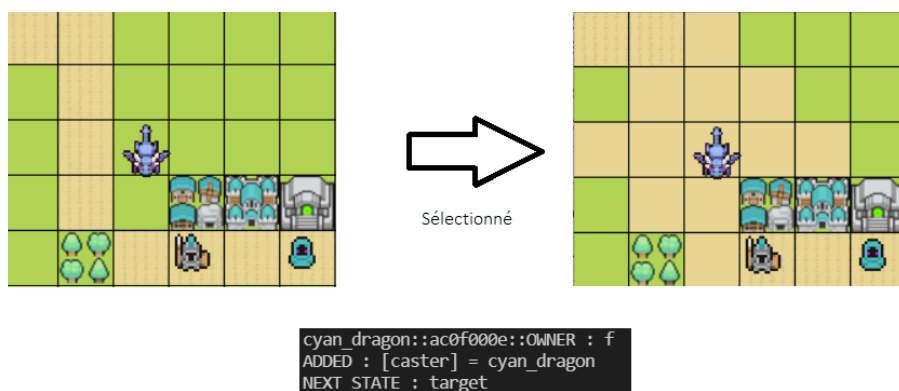


Figure 30 : le démarche de sélection

4.1.4 Commande des actions

Les acteurs ont différentes actions possibles :

- Les personnages ont des actions de déplacement , d'attaque.
- Les bâtiments ont des actions de récolte et peuvent invoquer des personnages.

Toutes ces méthodes seront définies dans le fichier *Script_Actions.bhv* dans le dossier *src/client/scripts* comme ci-contre.

```
FUNCTION Move
CALL FUNC::CheckPosition 3 ARGS:1 ARGS:2
NEQUAL RET:CheckPosition 0 JMP:3
LTE ACTOR_MGR::ARGS:0.MP 0 JMP:2
PROPERTY ACTOR_MGR::ARGS:0.X ARGS:1
PROPERTY ACTOR_MGR::ARGS:0.Y ARGS:2
END FUNCTION

FUNCTION Attack
CALL FUNC::CheckPosition 3 ARGS:1 ARGS:2
EQUAL RET:CheckPosition 0 JMP:7
INT dmg = 0
INT hp = 0
ADD dmg ACTOR_MGR::ARGS:0.DMG
ADD hp ACTOR_MGR::RET:CheckPosition.HP
SUB dmg ACTOR_MGR::RET:CheckPosition.DEF
SUB hp dmg
PROPERTY ACTOR_MGR::RET:CheckPosition.HP hp
PRINT_INT hp
END FUNCTION

FUNCTION Invoke_cyan
CALL FUNC::CheckPosition 3 ARGS:1 ARGS:2
NEQUAL RET:CheckPosition 0 JMP:5
INT ap = 100
SUB ap ACTOR_MGR::ARGS:0.AP
GT ap 0 JMP:2
ACTOR cyan_base BUILDING_CYANRESSOURCE ACTOR_MGR ARGS:1 ARGS:2
END FUNCTION
```

Figure 31 : Regle.bhv

Notre principale devise durant le développement du jeu est la flexibilité du code. En utilisant un script, il est possible de changer les attributs des personnages, actions ou fonctionnalités sans changer directement le code en C++. Si on souhaite modifier les règles du jeu ou ajouter de nouvelles fonctionnalités, par exemple un coup critique, on peut le faire très simplement en modifiant le script.

Remarque : Les acteurs ne peuvent pas faire leurs réactions si ce n'est pas son tour. Nous pourrions utiliser la touche "T" pour changer le tour.

4.2 Changement autonome

Les changements autonomes sont appliqués à chaque création ou mise à jour d'un état, après les changements extérieurs. Ils sont exécutés dans l'ordre suivant :

1. Si l'un des bâtiment de type base (Keep) est détruit, c'est à dire le nombre de HP du bâtiment est égale à 0, on affiche "GAME OVER"
2. Nous mettons à jour les attributs des acteurs de joueurs en fonction des règles du jeu
3. Nous appliquons les règles de déplacement du joueur et l'IA
4. Si le joueur a gagné, on affiche "GAME WON"

4.3 Conception logicielle

La diagramme des classes pour le moteur se constitue des classes suivantes :

- La classe **SelectionHandler** : gère la sélection, lors d'une action, SelectionHandler remplit une commande avec les paramètres souhaités. Il verrouille la couche de sélection.

```

SelectionHandler
+Selection: inline std::vector<state::Manageable**>
+FilteredSelection: inline std::map<std::string, state::Actor*>
+SelectionMask: inline std::vector<std::string>
+SelectionState: inline std::string
+Behaviour: inline Script*
+Packet: inline std::pair<std::string, std::string>
+Add(m: state::Manageable**): void
+Remove(m: state::Manageable**): void
+OnMouseLeft(x: int, y: int): void
+OnMouseRight(x: int, y: int): void
+Trash(): void
+ProcessSelection(m: state::Manageable**): int
+PrintSelection(): void
+NetFormat(): std::string
+Flush(packet: std::string): void
+ChangeAction(new action: std::string): void

```

Figure 32: La classe SelectionHandler

- La classe **InputHandler** : cette classe propose des événements (souris + clavier), la classe Player est abonnée à cette classe

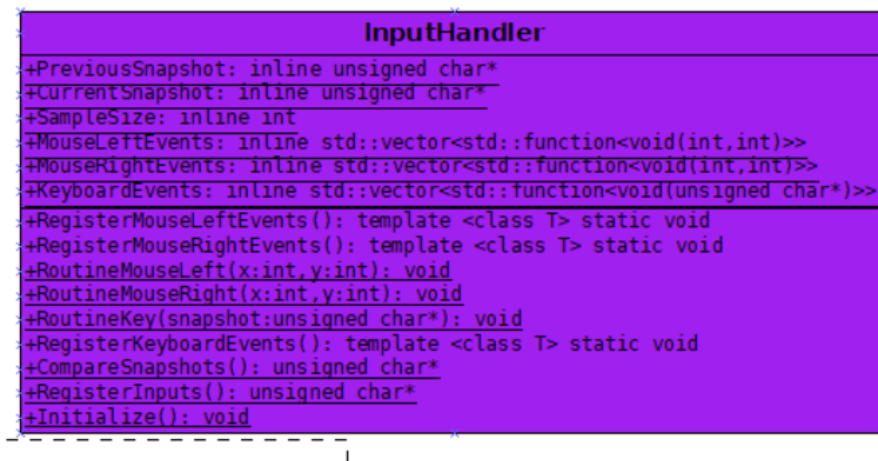


Figure 33: La classe InputHandler

- La classe **pattern** gère le motif, il génère la position des tiles à afficher.

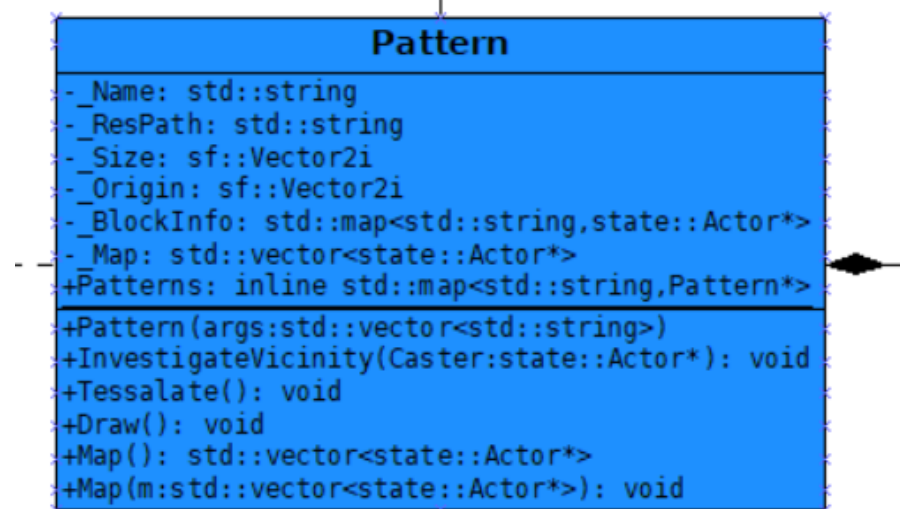


Figure 34: La classe Pattern

- La classe **NetCommand** : format des commandes à envoyer au moteur (dans le futur serveur) pour qu'il exécute une fonction située dans un script.

```

#NET_MSG,FORMAT
NET_CMD_MOVE,Move:$X;$X;$X,caster.ID;target.X;target.Y
NET_CMD_ATTACK,Attack:$X;$X;$X,caster.ID;target.X;target.Y
NET_CMD_INVOKE_CYAN,Invoke_cyan:$X;$X;$X,caster.ID;target.X;target.Y
NET_CMD_INVOKE_RED,Invoke_red:$X;$X;$X,caster.ID;target.X;target.Y
NET_CMD_TURN_FINISHED,OnTurnEnd,NONE

```

Figure 35: Le fichierNetMessage.csv

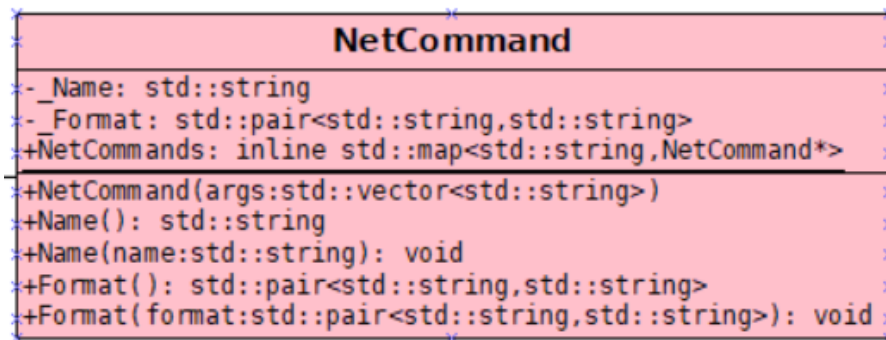


Figure 36: La classe NetCommand

- La classe **Action** : classe gérant les actions, par le biais d'un objet NetCommand , une action est liée à une fonction contenu dans un script.

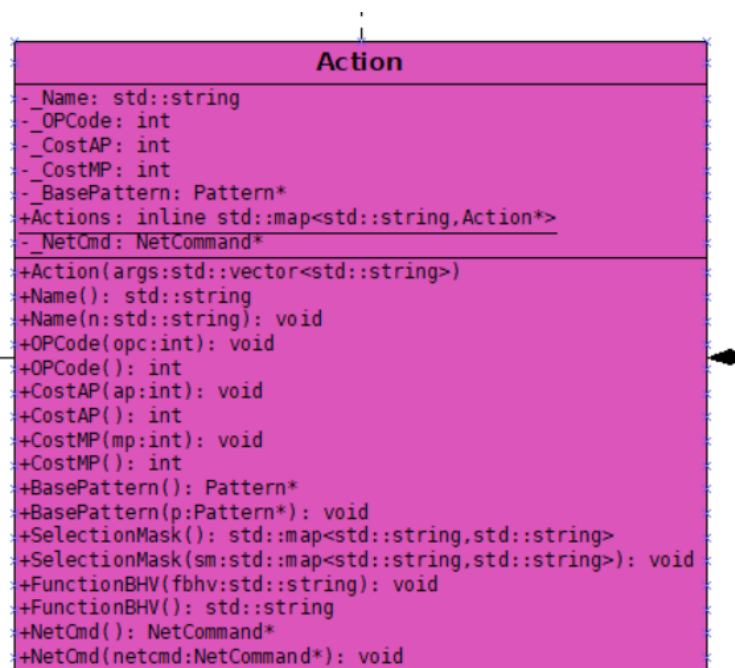


Figure 37: La classe Action

5 Intelligence Artificielle

5.1 Stratégie

5.1.1 IA aléatoire

Notre IA contrôle l'équipe rouge. Nous contrôlons l'équipe bleue. Au début de la partie, nous choisissons notre action. Une fois que nous terminons notre partie, nous appuyons sur la touche "T". Cela va passer le tour et va permettre à IA de jouer. L'IA va choisir entre différentes actions : se déplacer ou attaquer pour chaque personnage. Il ne peut attaquer que lorsqu'il y a des ennemis dans l'intervalle où il peut l'attaquer.

Pour chaque acteur, elle pourra effectuer autant d'actions que celle-ci a de points d'actions (expliqué en partie 4.1.1). Si elle épuise tous ses points, son tour se finira automatiquement. Selon l'action effectuée, notre IA disposera de moins d'actions à choisir parmi toutes celles disponibles pour rester cohérent avec nos règles de jeu. Par exemple si elle décide d'attaquer avec un acteur elle ne pourra pas se déplacer avec celui-ci ou attaquer de nouveau avec lui donc il ne lui restera plus que 3 choix.

Remarque : Nous avons changé une règle : quand un acteur d'une équipe est mort, le jeu est terminé. La première équipe à réussir à tuer l'acteur en premier gagne.

5.1.2 Comportement

On scripte le comportement de l'IA dans un fichier

```
PLAYER PLAYER_2 0 MNK  
PLAYER PLAYER_1 1 RandomAI
```

Le comportement de PLAYER_1 est dans le script "RandomAI"

```

FUNCTION TurnBegin
CALL STD_AI_BHV red_spearman.ID
CALL STD_AI_BHV red_bowman.ID
CALL STD_AI_BHV red_mage.ID
CALL STD_AI_BHV red_knight.ID
CALL STD_AI_BHV red_dragon.ID
CALL FUNC::EndTurn PlayerID
END FUNCTION

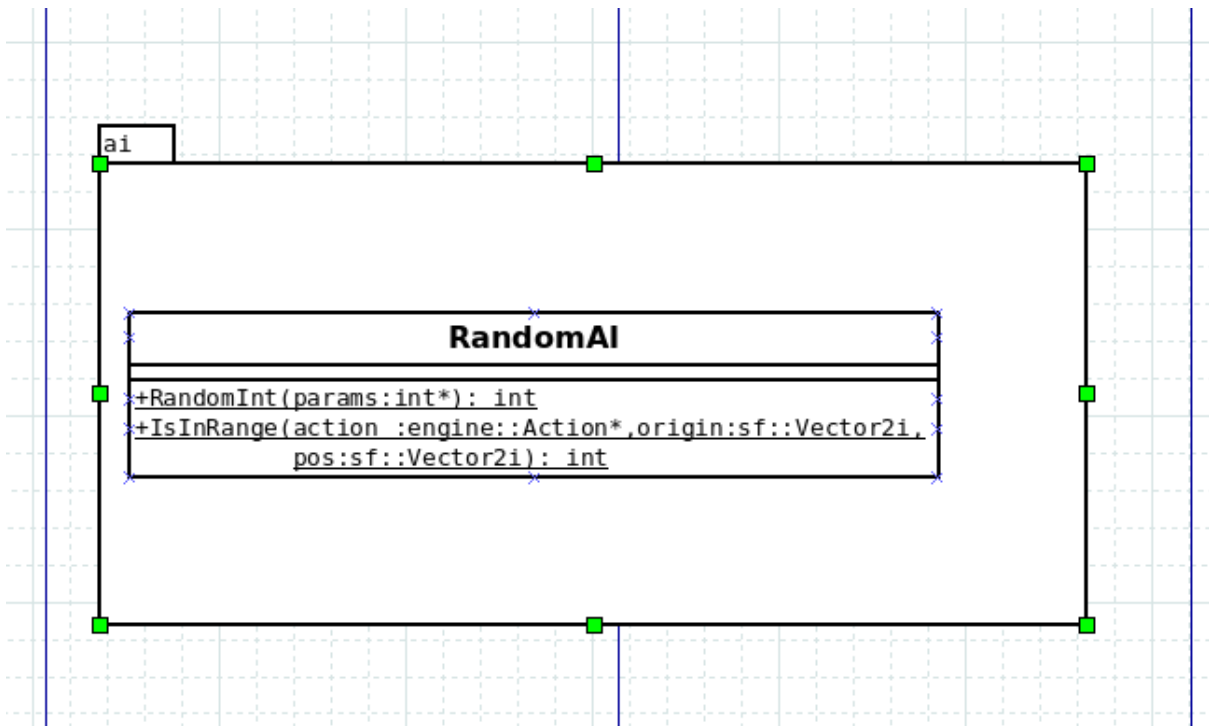
FUNCTION STD_AI_BHV
CALL FUNC::RandomInt -2 2
INT r0 = 0
ADD r0 RET:RandomInt
CALL FUNC::RandomInt -2 2
INT r1 = 0
ADD r1 RET:RandomInt
ADD r0 ACTOR_MGR::ARGS:0.X
ADD r1 ACTOR_MGR::ARGS:0.Y
CALL FUNC::CheckPosition 3 r0 r1
EQUAL RET:CheckPosition -1 JMP:6
EQUAL RET:CheckPosition 0 JMP:2
SEND STD_ATTACK ACTOR_MGR::ARGS:0.ID ACTOR_MGR::RET:CheckPosition.X ACTOR_MGR::RET:CheckPosition.Y
NEQUAL RET:CheckPosition 0 JMP:3
SUB r0 ACTOR_MGR::ARGS:0.X
SUB r1 ACTOR_MGR::ARGS:0.Y
SEND STD_MOVE_RELATIVE ACTOR_MGR::ARGS:0.ID r0 r1
END FUNCTION

```

Au début de chaque tour, la classe `state::WorldHandler` appelle la fonction `TurnBegin` de chaque script. Ainsi quand vient le tour de l'IA sa fonction `TurnBegin` est appelée et les actions sont effectuées.

5.2 Conception logicielle

Nous avons créé le diagramme des classes suivant :



La classe RandomAI propose des fonctions statiques que nous appelons dans le script.