

Rapport de jalon de Projet Logiciel Transversal

20 octobre 2016

Table des matières

1	Objectif	2
1.1	Présentation générale	2
1.2	À propos du jeu	2
1.2.1	Règles du jeu	2
1.2.2	Description générale	2
1.2.3	Description détaillée	2
2	Description et conception des états	8
2.1	Description des états	8
2.1.1	Etat Général	8
2.1.2	Etat éléments mobiles	8
2.1.3	Etat éléments fixes	9
2.1.4	Etat mode de jeu	9
2.1.5	Diagramme des classes d'état	13
3	Rendu	14
3.1	Stratégie de rendu d'un état	14
3.2	Conception logiciel	14
3.3	Exemple de rendu	17

1 Objectif

1.1 Présentation générale

L'objectif de ce projet est la réalisation d'un jeu de plateforme inspiré de "Dofus", avec des règles et des designs simplifiés

1.2 À propos du jeu

1.2.1 Règles du jeu

Ce jeu possède deux modes : un mode exploration et un mode combat. Le joueur se déplace librement sur la carte en mode exploration et peut passer un mode combat en décidant d'affronter un ennemi présent sur cette même carte en cliquant dessus. La finalité de ce jeu d'aventure est de combattre des ennemis afin de gagner en expérience et de là augmenter de niveau.

1.2.2 Description générale

Voici la description générale du jeu à réaliser :

1. Deux modes :
 - (a) Mode exploration
 - (b) Mode combat
2. Des acteurs jouables et non jouables
 - (a) Deux personnages (féminin, masculin)
 - (b) Cinq créatures (ennemis)
3. Six plateformes
 - (a) Trois cartes pour le mode exploration
 - (b) Trois cartes pour le mode combat (générées à partir des cartes d'exploration)
4. Trois menus
 - (a) Démarrer (Nouvelle partie, création de Personnage ...)
 - (b) Pause (Quitter, sauvegarder ...)
 - (c) Infos Personnage

1.2.3 Description détaillée

Modes :

- Exploration : Dans ce mode le joueur peut se déplacer librement sur la carte en cliquant sur une case, lancer un combat en se dirigeant vers une créature présente sur la carte.

- Combat : Dans ce mode il s'agit d'un duel entre le joueur et une adversaire (créature/monstre). Ce mode prend fin lorsqu'un des acteurs meurt.

Voici la description détaillée du jeu à réaliser :

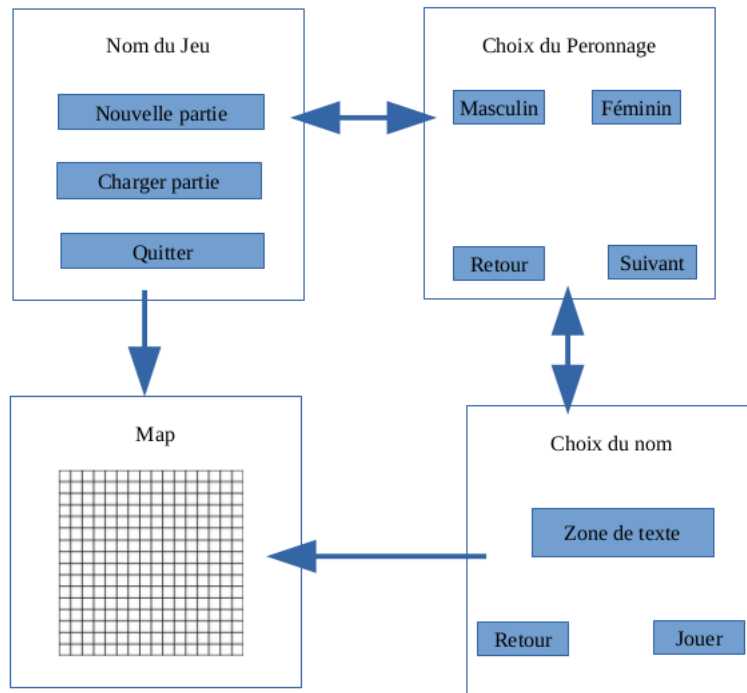


FIGURE 1 – Organisation et maquettes des menus

Au lancement du jeu, le menu démarrer s'affiche et trois options sont disponibles. Le bouton "Nouvelle partie" permet de lancer une nouvelle partie et redirige l'utilisateur vers un menu nommé "Choix du personnage". Le bouton "Charger partie" permet de lancer une partie enregistrée auparavant et redirige l'utilisateur vers la carte du jeu. Le bouton "Quitter" permet de mettre fin à l'exécution du jeu. Dans le menu "Choix du personnage", deux choix s'offrent à l'utilisateur soit un personnage masculin ou soit un personnage féminin. Une fois cette sélection effectuée l'utilisateur est dirigé vers un menu nommé "Choix du nom" permettant d'attribuer un pseudo au personnage retenu. Enfin, le bouton "Jouer" présent dans ce menu permet d'accéder à la plate-forme de jeu.

Menu Pause :

Au cours du jeu (hormis en mode de combat) via le bouton 'échap' du clavier, l'utilisateur peut accéder au menu "Pause" qui propose à ce dernier

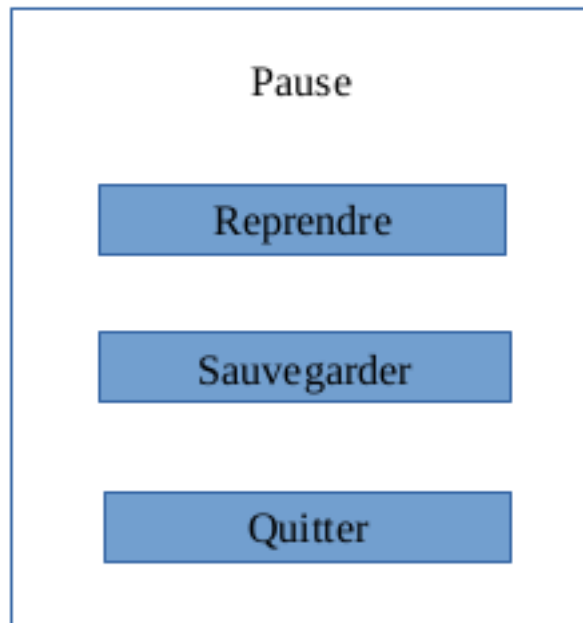
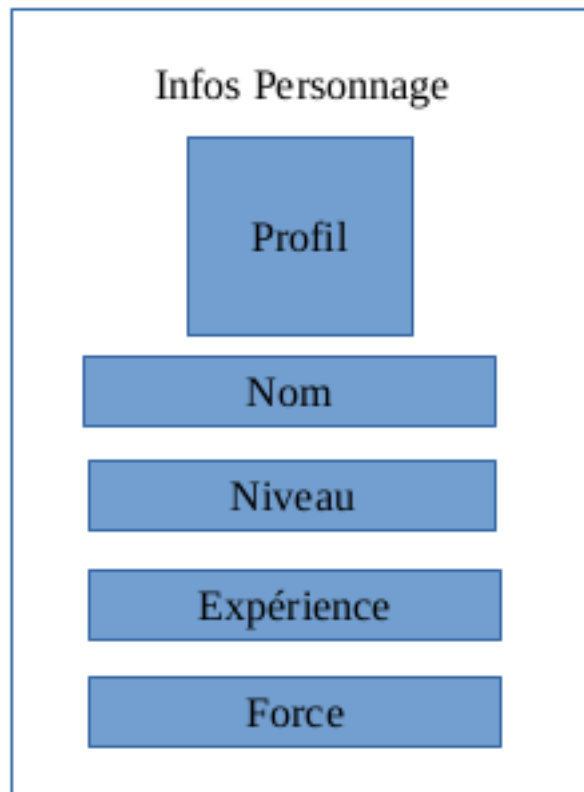


FIGURE 2 – *Menu de pause*

trois options : reprendre la partie du jeu en cours, sauvegarder la partie de jeu en cours et quitter le jeu en cours.

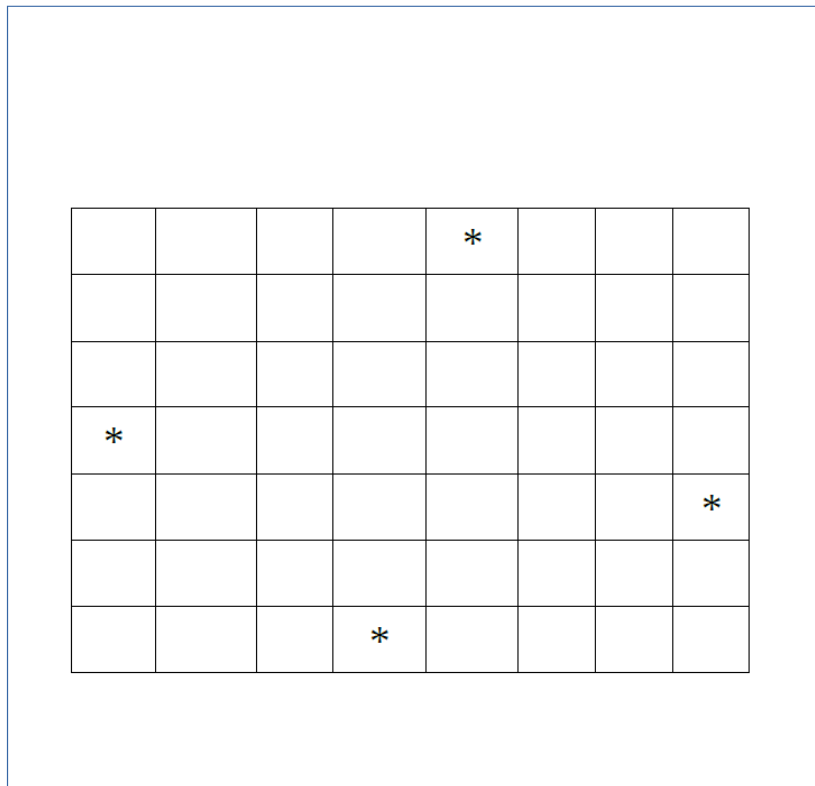
Menu Infos personnage :

FIGURE 3 – *Menu Infos*

Au cours du jeu (hormis en mode de combat) via le bouton ‘tab’ du clavier, l'utilisateur peut accéder au menu “Infos Personnage” qui contient tous les informations (nom, niveau, expérience, force, profil) sur le personnage du joueur.

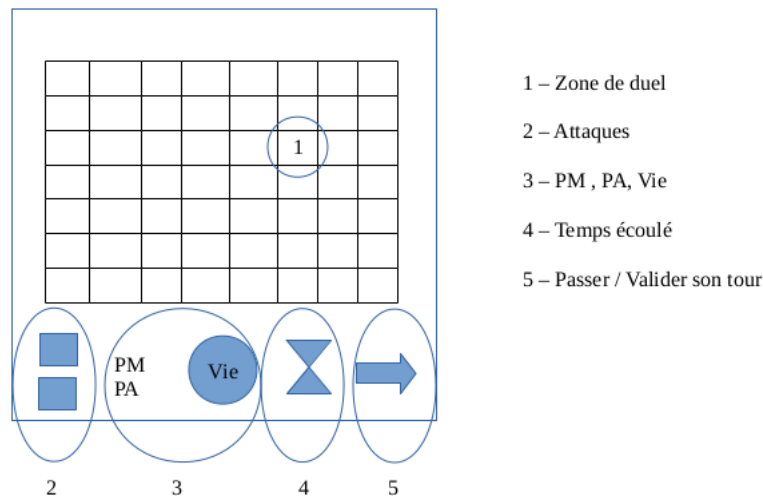
Cartes :

Deux types de cartes sont présents dans ce jeu : trois cartes dédiées au mode exploration et trois cartes dédiées au mode combat.

FIGURE 4 – *Modèle de carte en navigation*

- Le changement de carte s'effectue via les points d'accès (*)
- Sur la carte exploration, il y a présence du personnage (joueur) et des ennemis (créatures)

Carte de combat :

FIGURE 5 – *Modèle de carte en combat*

Sur la carte combat, il y a deux principales zones :

- Une zone de combat :
 - Dimension de la zone de duel : 12 carreaux * 10 carreaux
 - Effets visuels : Assombrissement du fond et zoom sur la zone de combat
 - Effets sonores : changement de musique
- une zone commandes/actions :
 - une attaque corps à corps
 - une attaque à distance
 - Points de Mouvement (nombre de cases pouvant être parcourues par tour), Points d'Action (nombre d'attaques pouvant être lancées par tour) et Points de Vie dépendent du niveau du joueur
 - Temps de jeu par tour de 60 secondes
 - Touche "Passer/Valider" permet de mettre fin à son tour

Les différents acteurs présents dans ce jeu sont :

- Deux personnages : un masculin et un féminin
- Cinq créatures (I.A)

Le joueur augmente de niveau en faisant des duels et en gagnant de l'expérience et les créatures ont des niveaux différents allant de 1 à 10.

Evolution du personnage :

Niveau	Vie	PM	PA	Exp
1	100	3	6	100
2	150	4	7	200
3	200	5	8	300
4	250	6	9	400
5	300	7	10	500
6	350	8	11	600
7	400	9	12	700
8	450	10	13	800
9	500	11	14	900
10	550	12	15	1000

Ce tableau résume l'évolution des paramètres Vie, PM, PA et Exp en fonction du niveau du personnage.

2 Description et conception des états

2.1 Description des états

Notre jeu consiste principalement au déplacement d'un personnage sur une carte et à l'affrontement de monstres sur cette même carte. Le jeu évolue d'un état à un autre et cela de manière continue en fonction des actions du joueur. Ces différents états sont décrits de manière globale dans les paragraphes suivants.

2.1.1 Etat Général

Un état de jeu est formée par un ensemble d'éléments mobiles (heros, monstres) et d'un ensemble d'éléments fixes (cases "vide", cases "accés"). Tout élément est défini par :

- Sa position (coordonnées x et y)
- Son identifiant TypeID qui permet de distinguer la nature (la classe) de l'élément

2.1.2 Etat éléments mobiles

Héros jouables et monstres sont définis dans deux classes héritant de trois classes chacune, pour définir leurs caractéristiques dans un système flexible mais néanmoins résilient au changement. Tout d'abord la classe "**Élément**" définit la position actuelle, selon x et y, de l'élément concernée. Cette classe permet également l'inclusion dans les listes définies par la classe "**ListeElements**", qui servira à l'enregistrement de tous les éléments de jeu. Ensuite la classe "**Mobile**" permet de définir la direction

actuelle de déplacement de l'élément, essentielle pour la gestion des sprites par le package de rendu. Enfin la classe "**Personnage**" comporte les principales caractéristiques des être vivants du jeu : Niveau, Vie, Force, Points d'action, Points de Mouvement, Attaque à distance et Attaque au corps à corps. Ces dernières sont définies par une autre classe "**Attaque**" qui peut être instanciée pour définir les dégâts infligés par une attaque ou le coût en point d'action de l'attaque. À part ces éléments communs, on notera que la gestion de l'expérience a motivé la distinction entre les classes "**Heros**" et "**Monstre**"

2.1.3 Etat éléments fixes

Les déplacements sur une carte s'effectuent de case en case d'une grille, et le changement de carte s'effectue sur des points d'accès disséminés de part et d'autre de la carte. Ces notions sont retranscrites dans l'existence d'une classe "**Grille**" contenant des éléments soit "vide" soit des points d'accès vers une carte suivante. Cette grille est composée d'une liste d'éléments et est représentée par la classe "**Grille**" qui hérite de "**ListeElements**", mais qui ne sera composée que de "**Statique**", classe héritant de "**Element**". La séparation en deux classes "**Vide**" et "**Acces**" permet de rajouter un attribut vers lequel l'élément non statique pointe.

2.1.4 Etat mode de jeu

Un élément important du gameplay résidant dans la distinction entre le mode "combat" et le mode "exploration", la classe "Combat" définit les principaux états propres au mode "combat" : l'existence d'un ordre de tour de jeu entre les différents personnages et d'un timer limitant la durée d'un tour. En outre, un attribut "EnCombat" contenue par la classe "**Etat**" indique si les joueurs se trouvent actuellement en combat ou en exploration.

Classe	Rôle
Etat	<p>Cette classe permet de définir un état de jeu.</p> <p>Attribut(s) : personnages, grille, combat, enCombat, visiteur, mapActuel</p> <p>Méthode(s) : Etat(), Etat(), getStatutGrille(), getPersonnage(), getRefPersonnage(), loadGrille(), getGrille(), getEnCombat(), setEnComabt(), rajouterPerso(), enleverPerso(), getRefCombat(), getMapActuel(), setMapActuel(), getPerso(), getPersoSize(), getGrilleSize(), getTile()</p>
ListeElements	<p>Cette classe permet de définir un objet de type “ListeElements”. Cette liste permet d’enregistrer tous les éléments mobiles et statiques nécessaires à l’état du jeu.</p> <p>Attribut(s) : elements, factory</p> <p>Méthode(s) : ListeElements(), ListeElements(), size(), getElement(), setElement(), isPerso(), ajoutElement()</p>
GrilleElements	<p>Cette classe permet de définir une liste d’éléments de type “GrilleElements”. Elle hérite de la classe “ListeElements”. Cette classe permet de construire une map de taille donnée (longueur * largeur) et d’y accéder aux cases sous forme matricielle. Pour faciliter le traitement, les cases sont enregistrées sous forme de liste d’éléments.</p> <p>Attribut(s) : longueur, largeur</p> <p>Méthode(s) : GrilleElements(), getLongueur(), getLargeur(), isAcces(), setCase(), charger(), setLongueur(), setLargeur(), ajoutCaseAcces()</p>
Combat	<p>Cette classe permet de définir un objet de type “Combat”. Elle gère la gestion d’un combat (fonctionnement tour par tour).</p> <p>Attribut(s) : timerDebutTour, listeTour, tourActuel</p> <p>Méthode(s) : combat(), Combat(), createListe(), tourSuivant(), getTimerDebutTour()</p>
ElementFactory	<p>Cette classe permet de définir un objet de type “ElementFactory” comme défini dans les standards de Design Pattern.</p> <p>Attribut(s) : timerDebutTour, listeTour, tourActuel</p> <p>Méthode(s) : combat(), Combat(), createListe(), tourSuivant(), getTimerDebutTour()</p>
IElementAlloc	<p>Cette classe abstraite permet de définir un objet de type “IElementAlloc”.</p> <p>Méthode(s) : IElementAlloc(), newInstance()</p>

HerosAlloc	Cette classe permet de définir un objet de type “HerosAlloc”. Elle hérite de la classe “IElementAlloc”. Attribut(s) : id Méthode(s) : ElementAlloc(), newInstance()
MonstreAlloc	Cette classe permet de définir un objet de type “MonstreAlloc”. Elle hérite de la classe “IElementAlloc”. Attribut(s) : id Méthode(s) : ElementAlloc(), newInstance()
VideAlloc	Cette classe permet de définir un objet de type “VideAlloc”. Elle hérite de la classe “IElementAlloc”. Attribut(s) : id Méthode(s) : ElementAlloc(), newInstance()
AccesAlloc	Cette classe permet de définir un objet de type “AccesAlloc”. Elle hérite de la classe “IElementAlloc”. Attribut(s) : id Méthode(s) : ElementAlloc(), newInstance()
IVisiteur	Cette classe abstraite permet de définir un objet de type “Ivisiteur” selon le modèle du Pattern Visiteur qui permet d’accéder à des informations via des pointeurs. Attribut(s) : pHeros, pVide, pAcces, pMonstre, lastType Méthode(s) : visiter()
Visiteur	Cette classe permet de définir un objet de type “Visiteur”. Elle hérite de la classe “Ivisiteur”. Méthode(s) : Visiter(), getpHeros(), getpMonstre(), getpAcces(), getpVide()
IAccepteVisite	Cette classe abstraite permet de définir une interface du type “IaccepteVisite”. Méthode(s) : accepte()
Element	Cette classe permet de définir un objet de type “Element”. Cet élément est caractérisé par une position sur la grille de coordonnées (x;y) et d’un type d’identifiant selon le type énuméré «TypeID» qui permet d’identifier le type d’élément selon l’ID défini. Cette classe est incluse dans la classe «ListeElements» Attribut(s) : x, y, typeID, elemID Méthode(s) : Element(), Element(), getTypeID() , accepte() , getX(), getY(), setX(), setY(), getElemID(), setElemID()

Statique	Cette classe abstraite permet de définir un élément de type “Statique”. Elle est la classe mère des classes « Acces » et « vide » qui sont les deux éléments statiques du jeu. Méthode(s) : Statique(), isAcces() , accepte() , getTypeID() , isStatic(), getTile()
Mobile	Cette classe abstraite permet de définir un élément de type “Mobile”. Elle est la classe mère de « Personnage » qui représente les éléments mobiles du jeu. Attribut(s) : joueur, direction, timer, enDeplacement Méthode(s) : Mobile(), isJoueur() , accepte() , getTypeID() , isStatic(), getDirection(), setDirection(), getEnDeplacement(), setEnDeplacement, getTimer(), setTimer()
Personnage	Cette classe abstraite permet de définir un élément mobile de type “Personnage”. Elle hérite de la classe “Mobile”. Attribut(s) : typePersonnage, force, niveau, ptAction, ptMouvement, vie, attaqueDistance, attaqueCAC, etatPerso Méthode(s) : Personnage(), Personnage(), isJoueur() , accepte() , getTypeID() , getTypePersonnage(), getForce(), getNiveau(), getPA(), getPM(), getVie(), getAttaqueDistance(), getAttaqueCAC(), getEtatPerso(), setTypePersonnage(), setForce(), setNiveau(), setPA(), setPM(), setVie(), setAttaqueDistance(), setAttaqueCAC(), setEtatPerso()
Attaque	Cette classe permet de définir un objet de type “Attaque”. Attribut(s) : typeAttaque, degat, coutPA Méthode(s) : Attaque(), Attaque(), getCoutPA(), setCoutPA()
Vide	Cette classe permet de définir un élément fixe de type “Vide”. Elle hérite de la classe “Statique”. Méthode(s) : Vide(), isAcces(), getTypeID(), accepte()
Acces	Cette classe permet de définir un élément fixe de type “Acces”. Elle hérite de la classe “Statique”. Méthode(s) : Acces(), isAcces(), getTypeID(), accepte()
Heros	Cette classe permet de définir un élément mobile de type “Heros”. Elle hérite de la classe “Personnage”. Attribut(s) : experience Méthode(s) : Heros(), isJoueur(), getExp(), setExp(), accepte()
Monstre	Cette classe permet de définir un élément mobile de type “Monstre”. Elle hérite de la classe “Personnage” tous ses attributs. Méthode(s) : Monstre(), isJoueur(), getTypeID(), accepte()

2.1.5 Diagramme des classes d'état

Légende des couleurs :

- Vert : Correspond à l'ensemble des classes Etat et ListeElements ainsi que sa classe fille
- Rouge : Correspond à la classe Combat
- Marron : Correspond à l'ensemble des classes ElementFactory et IElementAlloc ainsi que ses classes filles selon un Pattern design Factory
- Orange : Correspond à la classe Element et ses classes fille Statique, Mobile et Personnage
- Violet : Correspond à la classe Attaque
- Bleu : Correspond à l'ensemble des classes définies selon le modèle du Pattern design Visiteur
- Jaune : Correspond aux dernières classes fille de la classe Element (Heros, Monstre, Vide et Acces)
- Blanc : Correspond à l'ensemble des classes de type Énumération

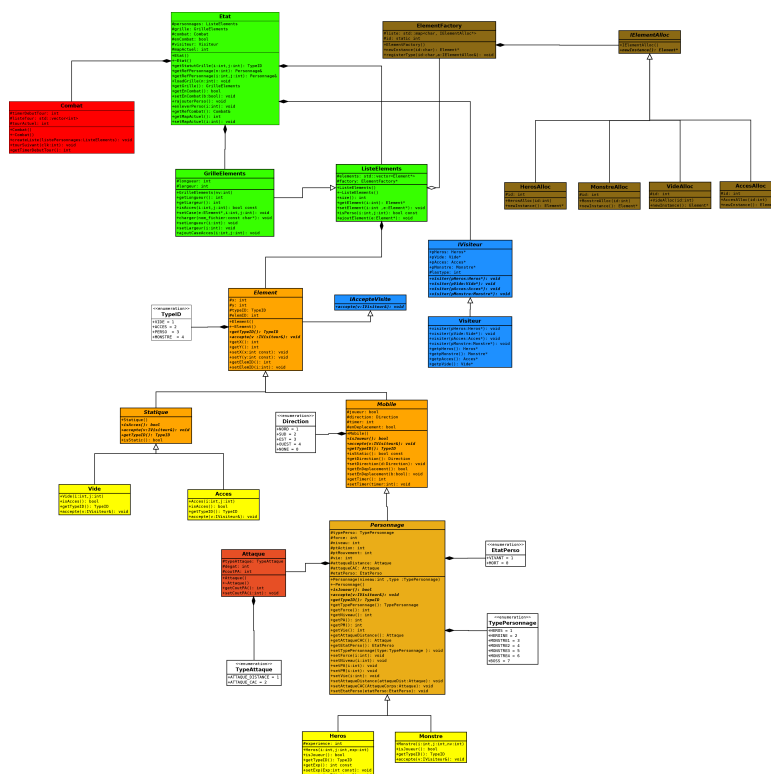


FIGURE 6 – *Diagramme de classe du package etat du projet*

3 Rendu

3.1 Stratégie de rendu d'un état

Une stratégie relativement simple est choisie pour réaliser le rendu d'un état du jeu. Cette stratégie consiste à segmenter le rendu en deux plans : un plan pour la carte (contenant tous les éléments statiques) et un second plan pour les éléments mobiles (personnages).

Une structure basée selon un Pattern Observer est utilisé afin de faire le lien entre le package State et le package Render. En effet, l'objectif de cette structure est d'observer l'état du jeu à rendre et d'apporter les changements survenus. Pour ce faire, l'implantation de six classes sont nécessaires :

Classes Observable et Observer : ces classes appartiennent au Pattern Observer et ont pour but d'observer les changements survenus sur l'état du jeu et de les notifier au rendu. Ces classes font le lien entre le package Etat et le package Rendu.

Classe Rendu : cette classe centralise les différents plan du rendu. Elle permet de mettre à jour le rendu d'un état de jeu. Elle est associée à différentes classes qui sont : RenduGrille, RenduPerso.

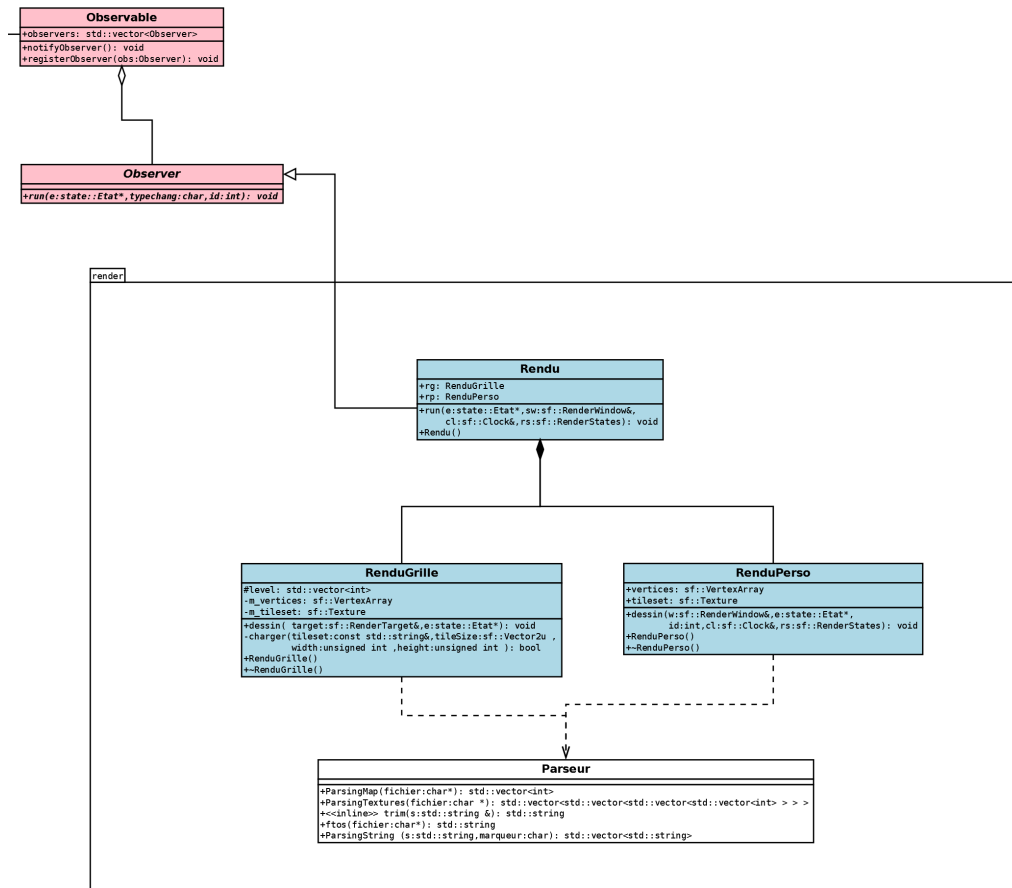
Classe Parseur : cette classe permet d'accéder aux différentes textures utilisées pour le rendu du jeu en passant par une lecture de fichiers .txt. Le chargement de ces textures se fait à l'aide de tableaux d'entiers qui sont ensuite exploités dans les classes du package Render.

3.2 Conception logiciel

Le tableau ci dessous permet de résumer l'ensemble des classes utiles et présentes pour définir un rendu d'un état de jeu ainsi que leur rôle respectif.

Nom	Role
Observable	Cette classe permet d'observer l'état du jeu. Attribut(s) : observers Méthode(s) : notifyObservers(), registerObserver()
Observer	Cette classe permet de mettre à jour le rendu en fonction des changements survenus dans l'état du jeu. Méthode(s) : run()
Rendu	Cette classe permet de gérer l'ensemble des plan du rendu à mettre à jour. Attribut(s) : rg, rp Méthode(s) : run(), Rendu()
RenduGrille	Cette classe permet de gérer le rendu de la carte du jeu (background). Attribut(s) : level, vertices, tileset Méthode(s) : dessin(), charger(), RenduGrille(), RenduGrille()
RenduPerso	Cette classe permet de gérer le rendu des personnages. Attribut(s) : vertices, tileset Méthode(s) : dessin(), RenduPerso(), RenduPerso()
Parseur	Cette classe permet d'accéder à la lecture d'un fichier .txt qui spécifie les textures pour les personnages et les cartes du jeu. Méthode(s) : parsingMap(), parsingTextures(), trim(), ftoS(), parsingString()

Diagramme de classe :

FIGURE 7 – *Package render*

3.3 Exemple de rendu



FIGURE 8 – *Exemple de rendu*