PROJET LOGICIEL

Auteurs : Fady ZAMBA-ZAMBA
Ramzi Mouad

par : GOSSELIN

ER Bernard

TRANSVERSAL

29 SEPTEMBRE 2016

Table des matières

1. Objectif

1. Présentation générale

Ce projet consistera à réaliser un jeu de combat de type JRPG (voir figure d'illustration).

1.2 Règles du jeu

Tout d'abord, le joueur commence son tour en choisissant une attaque. Une fois l'attaque choisie, le personnage la mettra en œuvre. L'adversaire (IA), suite à cela, va riposter à son tour en lançant une attaque. Il faut noter que le joueur pourra également utiliser des attaques spéciales (qui correspondent à des dommages plus importants) ; ces dernières ne pourront être réutilisées qu'après plusieurs tours. Les barres de vies détermineront ensuite le vainqueur (le personnage ayant une barre de vie vide sera considéré comme le perdant).

1.3 Nos ressources

Nous avons tout d'abord eu besoin de choisir des « Spritesheets » qui permettront par la suite de réaliser des animations sur les personnages.

Ensuite, il a fallu déterminer une image de fond, afin de représenter la zone environnante des deux combattants. Nous avons retravaillé l'image de fond afin que celle-ci s'anime lors des combats.

Enfin, nous avons rajouté quelques fichiers sonores afin de rendre le jeu plus vivant (musique d'introduction, bruits des coups donnés, etc…).

2. Description et conception des états

2.1. Description des états

Un état du jeu est formé par un ensemble d'éléments fixes (Background, Gauge, Time) et un ensemble d'éléments mobiles (Player).

2.1.1. Etat éléments fixes

Element fixe « Background ». Le background constituera l'environnement dans laquelle les deux combattants s'affrontent. C'est au joueur de choisir le décor Ainsi, il aura le choix entre plusieurs textures (un combat se déroulant dans une pièce, ou un combat se déroulant en plein air par exemple).

Element fixe « Gauge». Par dessus le Background se trouveront deux jauges:

- Jauge de vie : une zone rectangulaire initialisée par une couleur verte. A chaque coup subit, le rectangle diminuera en longueur(diminution des point des vie).

- Jauge d'expérience : une zone rectangulaire, mais cette fois-ci qui se charge en fonction du nombre de coups émis. Une fois la jauge chargée, le joueur pour activer une attaque surpuissante. Une fois l'attaque lancée, la jauge se videra. (le joueur devra réitérer les mêmes actions pour recharger la jauge).

Element fixe «Time». Il y'aura ici un emplacement dédié au chronomètre afin de limiter la partie. Il faut savoir que deux combattants peuvent ne pas attaquer et ainsi se mettre en position défensive : le jeu durerait trop longtemps s'il n'y avait que des défenses lancées de chaque côté. Ainsi, à la fin du temps réglementaire, celui ayant le plus de point de vie sera considéré comme gagnant.

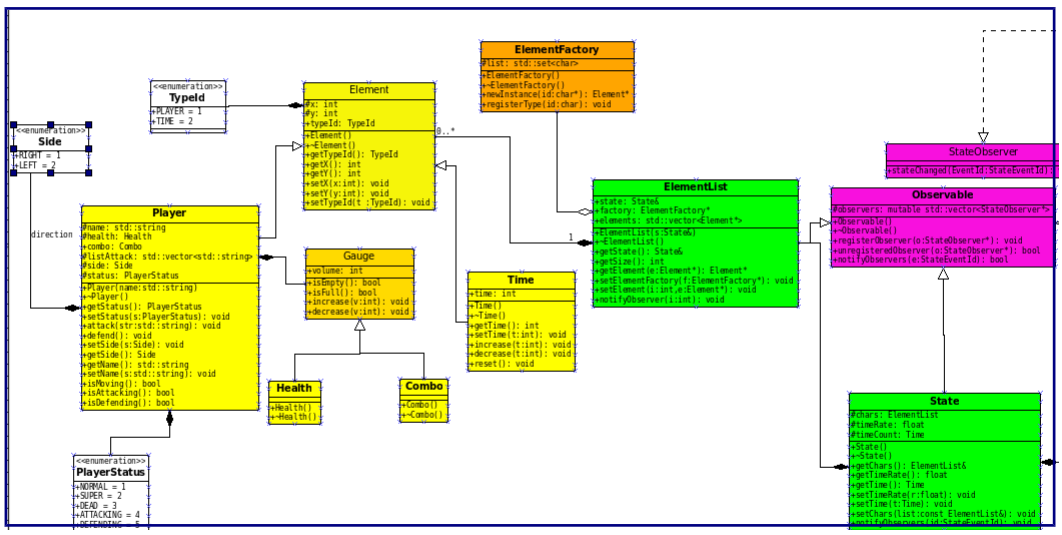
2.1.2 Etat élément mobile

Elément mobile « Player ». Cet élément est dirigé par le joueur. Il sera caractérisé par une énumération. En effet, on lui attribuera plusieurs « statuts » (**NORMAL, SUPER, DEAD,** etc…) permettant de déterminer l'état dans lequel se trouve le joueur et ainsi réaliser les méthodes adéquates.

2.2 Conception logiciel

2.2.1 Diagramme des classes

Le diagramme des classes pour les états est représenté ci-dessous.



2. Rendu : stratégie et conception

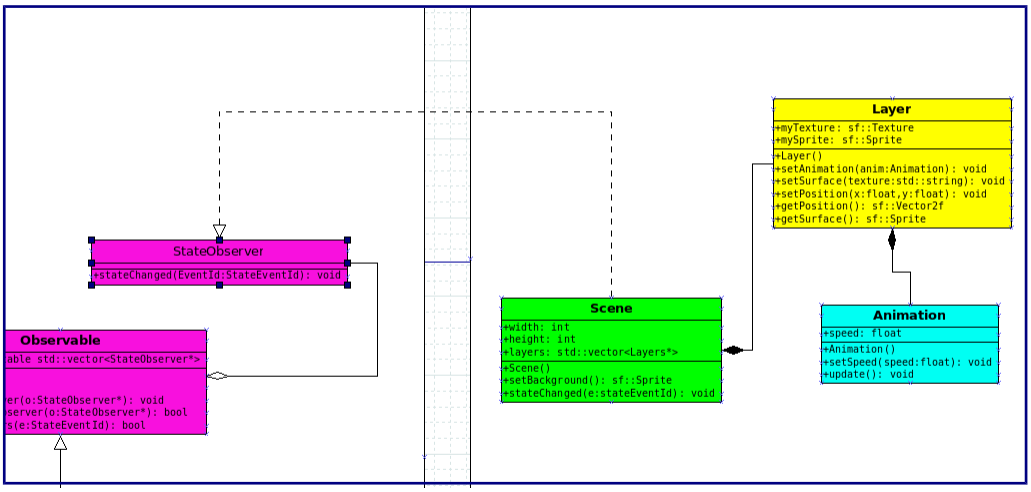
3.1. Stratégie de rendu d'un état

Le rendu d'un état sera délégué à un package spécifique « render ». Ce dernier contient toutes les classes don't le but est de gérer les textures relatives aux éléments du jeux. Nous décomposons ainsi ces élements en trois sous-catégories. Les deux joueurs (droite et gauche) , le background, qui pour l'instant restera statique, et les informations (bars de vies, bars de supers attaques – Combo et le temps).

Le package du rendu est relié à l'état par le biais de l'observateur stateObserver. Ce dernier permet de détecter les changement qui ont été produits au nivau des éléments du jeu et qui engendrent donc un changement d'état.

3.2. Conception logiciel

Le diagramme des classes pour le rendu, est illustré ci-dessous.



Scène. La classe *Scène* permet de gérer le bon déroulement des couches de rendu sur l'écran. Elle fera appel aux instances de la classe *Layer* afin de pouvoir organiser les différentes actions à réaliser lors d'un changement d'état (package state). La méthode ***stateChanged*** permettra, graçe à différents switch, de sélectionner le layout à éxécuter, et ainsi, de sélectionner l'animation à réaliser.

Layer. Cette classe représente la couche permettant de réaliser un rendu visuel. Nous utiliserons la librairie SFML. Nous allons ainsi pouvoir charger l'emplacement de nos images et y mettre nos textures; nous allons pouvoir créer nos sprites à partir de ces mêmes textures et pouvoir également changer la position des sorites sur l'écran.

Animations. Toute animation sera instanciée par cette classe. Nous avons fait ce choix car nous allons posséder plusieurs types d'animation:

- animation sans mouvement : lorsque le personnage est dans l'état attente/repos, nous allons faire appel à une animation fixe (haussement des épaules/ respiration)
- animation avec mouvement: lorsque le joueur aura choisi d'attaquer ou de réaliser un mouvement, nous allons faire appel à des sprites (d'où la nécessité de la classe Layer) prédéfinies dans une liste afin de ne sélectionner que ce qui nous intéresse.