École polytechnique de l'Université de Nantes
Département informatique

Projet Bomberman

Cahier des charges technique - Conception

Bisiaux Alexandre *(Relecteur)*
Guihal Maxime *(Rédacteur)*
Guillermic Brice *(Rédacteur)*
Rousseau Simon *(Relecteur)*

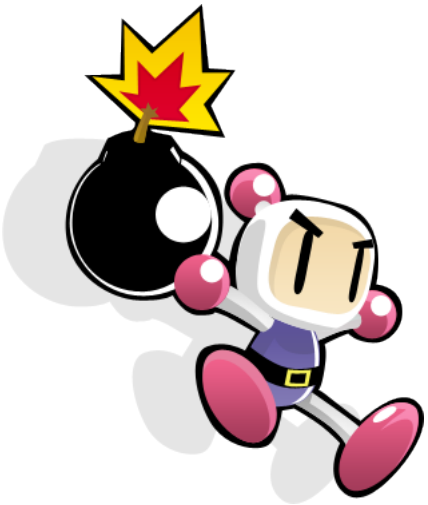


Table des matières

1 Structure du programme

Ce document présente la conception choisie pour le programme. Celui-ci est découpé en 8 modules distincts pouvant être développés séparément. La structure globale du programme sera d'abord fournie puis chacun des modules sera décrit par un diagramme de classes UML et accompagné d'explications pour expliciter leurs interfaces.

1.1 Bibliothèques utilisées

Ce programme sera réalisé à l'aide du langage **C++**. La bibliothèque graphique utilisée sera la **SFML 1.6** [1] choisie pour sa simplicité et son efficacité. Celle-ci est sous licence zlib/png qui permet son utilisation sans aucune contreparties.

1.2 Les composants du programme

La structure choisie pour le programme permet à celui-ci de se décomposer en huit composants (modules) indépendants, pouvant être développés parallèlement, dont voici les caractéristiques :

Module de gestion du menu Ce composant est le premier à être appelé par le programme. Il permet de gérer l'enchaînement des pages de menu pour configurer le programme et pour créer ou rejoindre une partie.

Module de gestion de l'interface de jeu Appelé par le menu lorsqu'une partie commence, ce composant permet de gérer l'affichage du plateau en fonction des informations reçues par le moteur de jeu.

Module de gestion des contrôleurs Ce composant permet l'interaction avec les contrôleurs de jeu (clavier, gamepad, Wiimote, ...) en fonction des configurations, puis communique ces informations avec les autres composants.

Module de gestion du réseau Ce module central du programme coordonne les autres composants en transférant les interactions avec les contrôleurs au moteur de jeu et le rendu du plateau de jeu à l'interface. Il gère aussi l'envoi et la réception des paquets sur un réseau local.

Module de gestion du jeu Ce composant est le moteur du jeu. Son rôle est d'analyser le plateau de jeu afin d'en déduire les mouvements à effectuer en fonctions des interactions avec les contrôleurs.

Module de gestion de la configuration Ce composant gère l'interaction avec les fichiers de configuration du jeu pour y stocker les touches paramétrés, et toutes les autres options du programme.

1. `http://www.sfml-dev.org`

Module de gestion des skins Le programme permettant de changer de skin, ce composant permet de charger les images et les autres éléments de l'interface en fonction du skin choisi.

Module de gestion du son Ce composant permet simplement de jouer des sons ou des musiques selon la configuration du programme au cours de son exécution.

1.3 Relations entre les composants

Afin de communiquer avec les autres composants, chaque module fournit une ou plusieurs interfaces, qui seront les seules parties publiques des composants, utilisable par les autres composants. La liaison des interfaces et des modules entre-eux sont représentés dans le diagramme de composants en annexe à ce dossier. Chacune des interfaces est détaillée dans la partie consacrée au module ci-après.

Afin d'avoir une vue globale sur le fonctionnement du programme, les relations entre les différents composants du programme ont été représentées sur un diagramme de déploiement présenté en annexe.

2 Module Menu

2.1 Interfaces fournies

Ce composant fournit deux interfaces aux autres modules :

IMenuToMain Cette interface permet de lier le composant à la méthode *main* d'entrée du programme. Elle fournit la méthode **run** qui renvoit une des constantes suivantes : EXITGAME ou EXITERROR selon que le programme s'arrête sans ou avec un erreur.

IMenuToGameInterface Cette interface permet de lier le menu au composant d'interface de jeu lorsque celle-ci appelle le menu au cours d'une partie (lors d'une pause). La méthode **runPause** est alors appelée, renvoyant un signal à l'interface permettant de déterminer si le joueur reprend le jeu, ou quitte le programme. Le joueur pris en paramètre correspond à celui qui a appuyé sur le bouton pause.

2.2 Interfaces requises

Pour fonctionner, le composant Menu a besoin de 5 interfaces :
- INetworkToMenu
- IControllerToMenu
- ISkin
- IConfigFile
- ISound

2.3 Structure du composant

Le module est composé d'un tableau de *IMenuScreen* pour chacun des menus proposés aux joueurs, qui s'appellent les uns les autres en renvoyant l'indice de l'écran suivant. Chaque écran est composé de plusieurs *widgets* dessinables et actionnables spécialisés selon leurs fonctions.

Le diagramme de classes représentant ce composant est fourni en annexe

3 Module Interface de jeu

3.1 Interfaces fournies

Ce composant fournit une interface aux autres modules :

IGameInterfaceToMenu Cette interface fournit la méthode **run** qui est appelée par le menu lorsqu'une partie commence. Elle prend en paramètres la fenêtre du programme, les scores de chaque joueur, et une référence vers le joueur gagnant. Elle renvoie une constante indiquant la fin de la partie (EXITGAME), ou une erreur (EXITERROR).

3.2 Interfaces requises

Pour fonctionner, le composant Menu a besoin de 4 interfaces :
- INetworkToGameInterface
- IMenuToGameInterface
- ISkin
- ISound

3.3 Structure du composant

Le module reçoit en boucle l'état du jeu par l'intermédiaire du module Réseau (INetworkToGameInterface), et affiche les différents éléments du plateau de jeu en fonction de cet état, ainsi que le score des différents joueurs. Tout comme les autres composants principaux, le gestionnaire de l'interface de jeu n'est instancié qu'une seule fois par exécution du programme.

Le diagramme de classes représentant ce composant est fourni en annexe.

4 Module Contrôleurs de jeu

4.1 Interfaces fournies

Ce composant fournit deux interfaces aux autres modules :

IControllerToNetwork Cette interface contient la méthode **getKeysPressed** qui renvoie l'état des touches pour chaque joueur de l'ordinateur, pour les transmettre ensuite au moteur de jeu.

IControllerToMenu Cette interface fournit un ensemble de méthodes permettant de contrôler le menu avec les manettes de jeu et le clavier (**getKeyPressed**, **getCharPressed**). Des méthodes permettent aussi de récupérer et de sauvegarder la configuration des touches pour chaque joueur (**getConfig**, **save**, **reloadConfig**), ainsi que pour en assigner de nouvelles (**setPlayerKey**, **setPlayerController**).

4.2 Interfaces requises

Ce composant n'a pas besoin d'interfaces avec d'autres modules pour fonctionner.

4.3 Structure du composant

La gestion des contrôleurs de jeu est effectuée au niveau de la classe ControllerManager. Cette classe se charge d'assigner à chaque joueur un contrôleur disponible, de renvoyer les actions émises par chacun des contrôleurs, de configurer les commandes de jeu pour chaque joueur. La classe Wiimote utilise l'interface CWii pour gérer les contrôleurs de type Wiimote.

Le diagramme de classes représentant ce composant est fourni en annexe.

5 Module Réseau

5.1 Interfaces fournies

Ce composant fournit trois interfaces aux autres modules :

INetworkToGameInterface Cette interface fournit une méthode **isPaused** qui renvoie le numéro du joueur qui a appuyé sur le bouton Pause, ou une valeur négative si le jeu n'est pas en pause, à l'interface graphique du jeu. Deux autres méthodes permettent de transférer le plateau de jeu (**getBoard**), ainsi que de donner l'état de la partie (si elle est finie ou non) (**isFinished**).

INetworkToGameEngine Cette interface fournit également une méthode **isPaused** qui renvoie cette fois-ci une valeur booléenne. Une seconde méthode **getKeysPressed** permet de transférer au moteur de jeu l'état des touches de chaque joueur de la partie, récupérées par l'interface IControllerToNetwork.

INetworkToMenu Cette interface fournit un ensemble de méthodes pour communiquer entre le menu et le réseau lors d'une création de partie, ou lorsqu'un joueur veut rejoindre une partie, afin de vérifier la connexion avec l'ordinateur hôte, de récupérer le nombre de places disponibles ainsi que les noms et les scores des autres joueurs. D'autres méthodes permettent de savoir si la partie est commencée, ou de la commencer.

5.2 Interfaces requises

Pour fonctionner, le composant Menu a besoin de deux interfaces :
- IControllerToNetwork
- IGameEngineToNetwork

5.3 Structure du composant

Ce module central du programme permet, pendant une partie, de faire communiquer le moteur de jeu avec les interfaces graphiques de chaque joueur, qu'ils soient locaux ou distants. Le composant demande en boucle l'état des touches des différents joueurs de la partie, les transmet au moteur de jeu par l'interface INetworkToGameEngine, puis récupère le plateau de jeu pour le transférer à l'interface graphique locale par l'interface INetworkToGameInterface, ou celles distantes en transmettant le plateau de jeu au module Réseau des autres ordinateurs si l'ordinateur est l'hôte de la partie, ou en transmettant à l'interface graphique locale le plateau de jeu reçu par le réseau si l'ordinateur est client.

Le diagramme de classes représentant ce composant est fourni en annexe.

6 Module Moteur de jeu

6.1 Interfaces fournies

Ce composant fournit une interface aux autres modules :

IGameEngineToNetwork Cette interface fournit des méthodes au module Réseau afin qu'il récupère le plateau de jeu (**getBoard**), l'état de la partie (**isFinished**), et pour définir la configuration de la partie (**setGameConfig**).

6.2 Interfaces requises

Pour fonctionner, le composant Moteur a besoin d'une unique interface :
- INetworkToGameEngine

6.3 Structure du composant

Ce module reçoit en boucle l'état des touches de chaque joueur de la partie par l'intermédiaire de l'interfae INetworkToGameEngine, puis à partir de ces informations, il calcule la position et l'état de chaque objet du plateau, puis le transfère au module Réseau. Chaque objet du plateau est défini dans sa propre classe qui est contenue dans la classe **Board** (le plateau), géré par le singleton **GameEngineManager**.

Le diagramme de classes représentant ce composant est fourni en annexe.

7 Module Configuration

7.1 Interfaces fournies

Ce composant fournit une interface aux autres modules :

IConfigFile Cette interface fournit toutes les méthodes utiles à la lecture (**getStringValue** et **getIntValue**) et l'écriture (**setStringValue** et **setIntValue**) dans les fichiers de configuration. Un constructeur avec le fichier de configuration en paramètre est également fourni.

7.2 Interfaces requises

Ce composant n'a pas besoin d'interfaces avec d'autres modules pour fonctionner.

7.3 Structure du composant

Ce module ouvre simple le fichier de configuration passé en paramètre pour y lire ou écrire des ensemble (clé, valeur). Toute modification au fichier par ce module est instantanément retranscrite dans le fichier.

Le diagramme de classes représentant ce composant est fourni en annexe.

8 Module Skin

8.1 Interfaces fournies

Ce composant fournit une interface aux autres modules :

ISkin Cette interface fournit un ensemble de méthode pour charger des images (**loadImage**) ou des couleurs (**getColor**) dans le programme, en fonction d'un skin. Des méthodes permettant de lister les skins (**getSkinsList**, **getSkin**), et de le changer (**setSkin**, **saveConfig**, **reloadConfig**) sont également disponibles.

8.2 Interfaces requises

Pour fonctionner, le composant Skin a besoin d'une interface :
– IConfigFile

8.3 Structure du composant

Ce module charge les images en fonction d'une clé donnée par l'énumération EImage dans le bon dossier de skin. Il permet aussi de lire des couleurs dans un fichier de configuration, en fonction d'une clé donnée par l'énumération EColorKey.

Le diagramme de classes représentant ce composant est fourni en annexe.

9 Module Son

9.1 Interfaces fournies

Ce composant fournit une interface aux autres modules :

ISound Cette interface fournit un ensemble de méthodes pour pouvoir jouer un son (**playSound**) ou une musique (**plysMusic**), d'arrêter une musique (**stopMusic**), ainsi que pour gérer le volume des sons et leur configuration (**saveConfig**, **reloadConfig**, etc...).

9.2 Interfaces requises

Pour fonctionner, le composant Son a besoin d'une interface :
– IConfigFile

9.3 Structure du composant

Ce module charge une musique ou un son en fonction d'une clé fournie par les énumérations ESound et EMusic, puis la joue en s'appuyant sur les classes audio de la bibliothèque SFML.

Le diagramme de classes représentant ce composant est fourni en annexe.

Conclusion

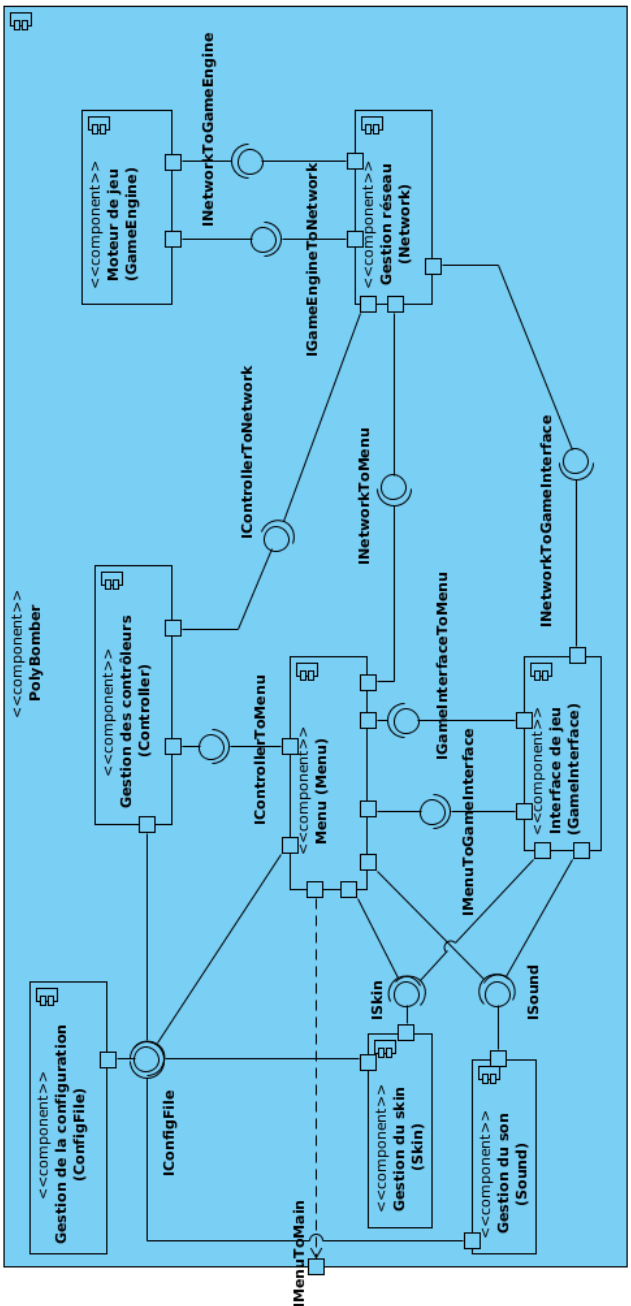
Cette description de chaque module est suivie en annexes des diagrammes de classes correspondants, ainsi que du diagramme de composants et de déploiement du programme.

Cette conception est susceptible d'évoluer au cours du développement du programme. Elle donne juste une vue globale de la structure du projet.

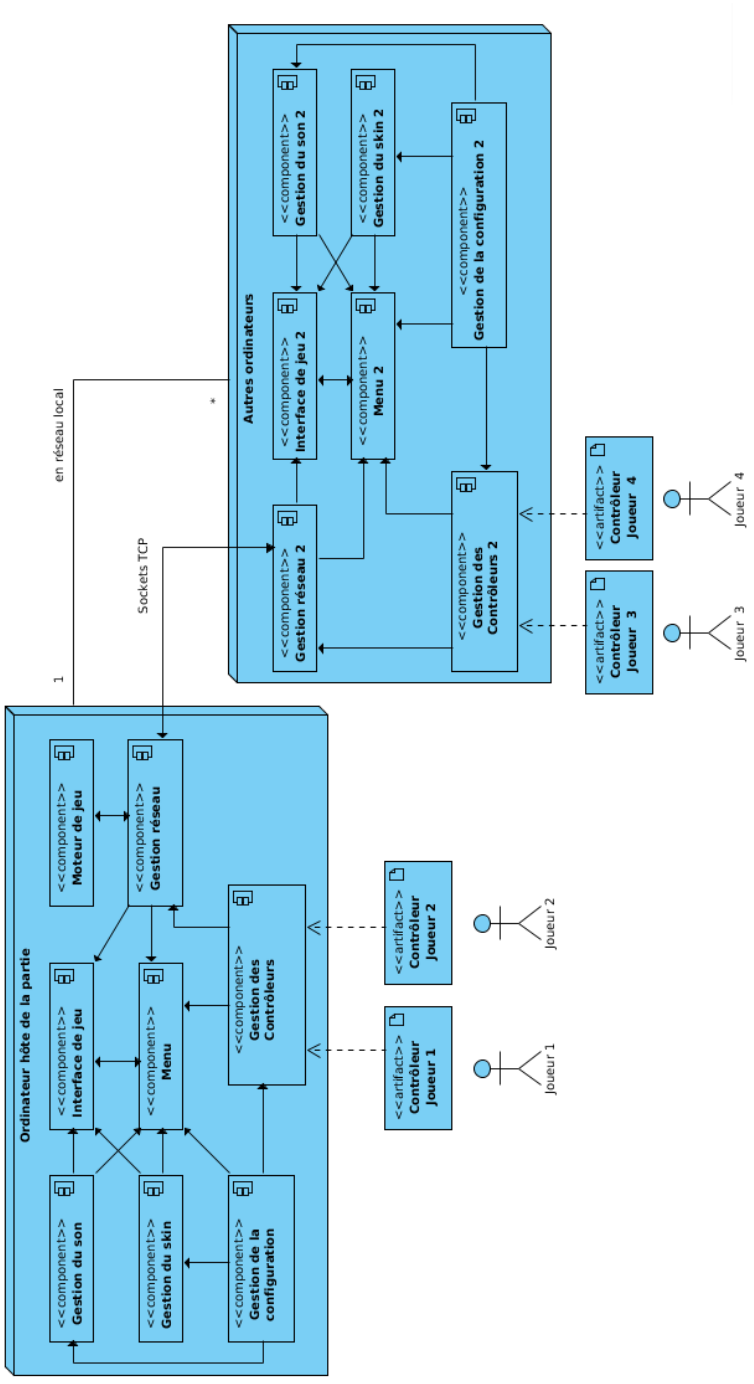
Pour les diagrammes de classes, la légende des couleurs est la suivante :
- **Bleu** : Classe
- **Vert** : Interface fournie
- **Orange** : Interface requise
- **Jaune** : Structure
- **Violet** : Enumération
- **Gris** : Classe déjà implémentée

Annexes

A Diagramme de composants

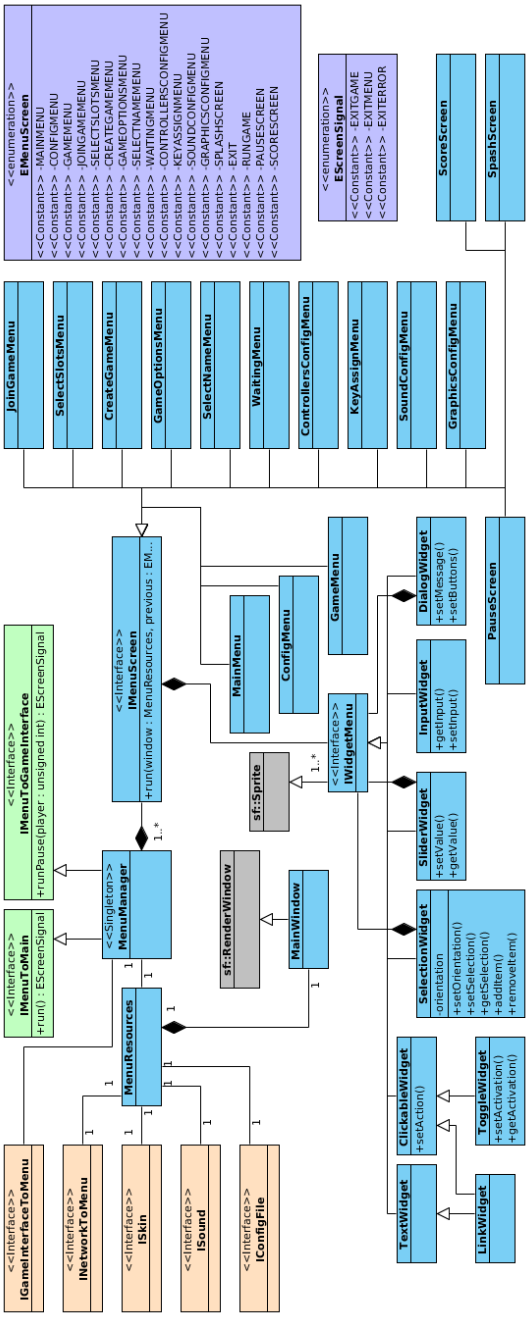


B Diagramme de déploiement

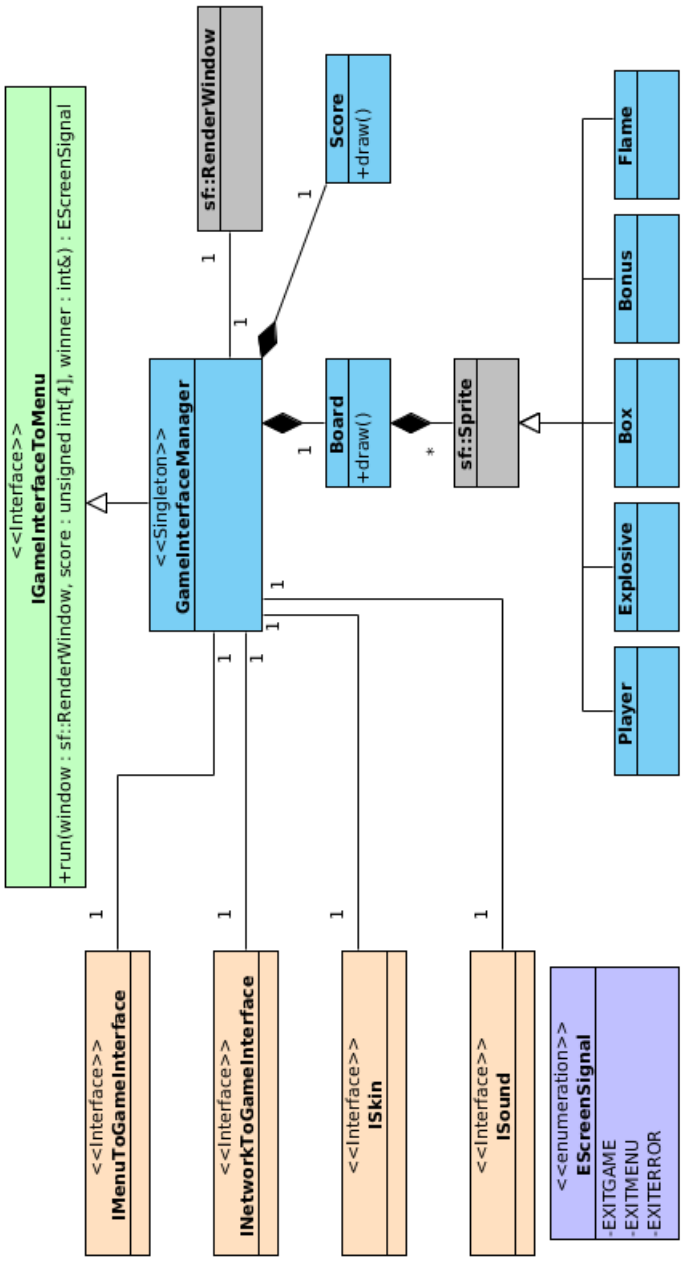


C Diagrammes de classes

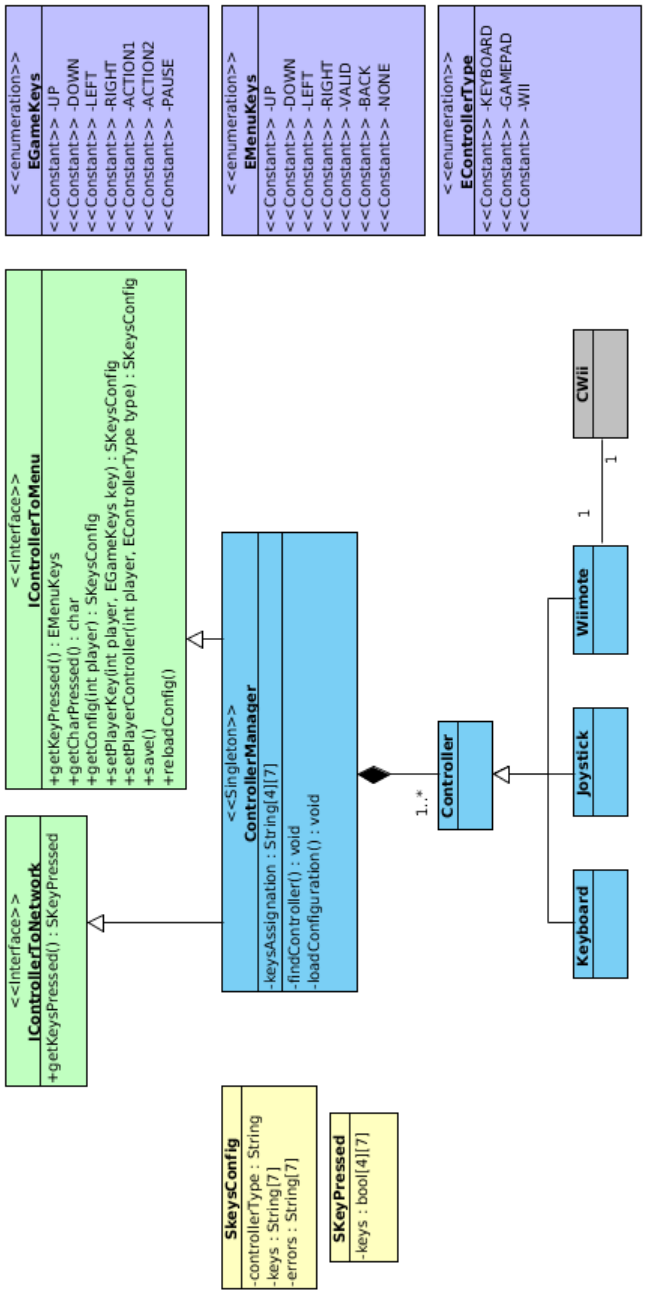
C.1 Module Menu



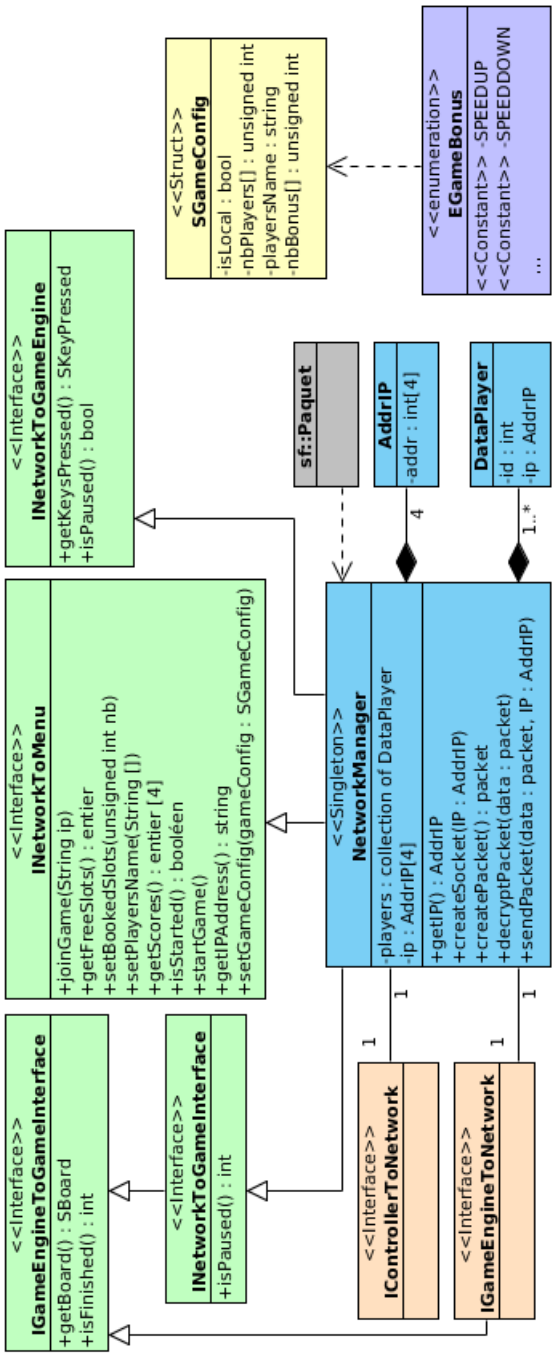
C.2 Module Interface de jeu



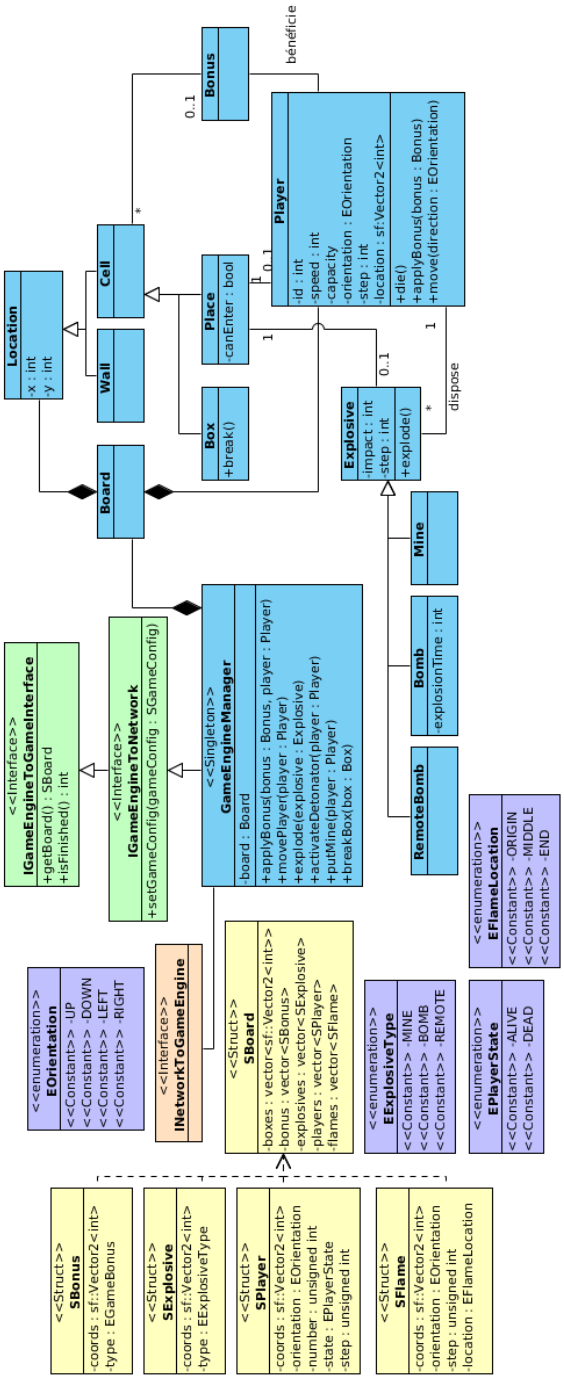
C.3 Module Contrôleurs de jeu



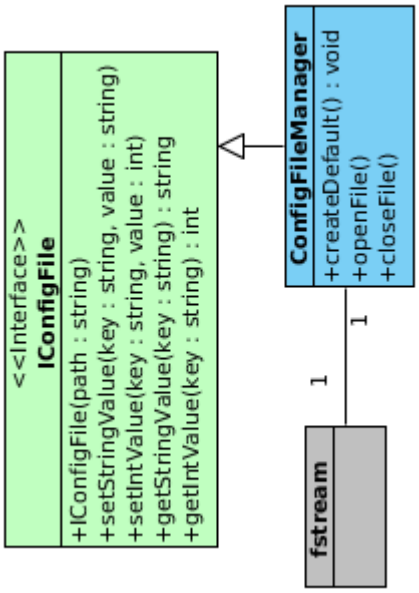
C.4 Module Réseau



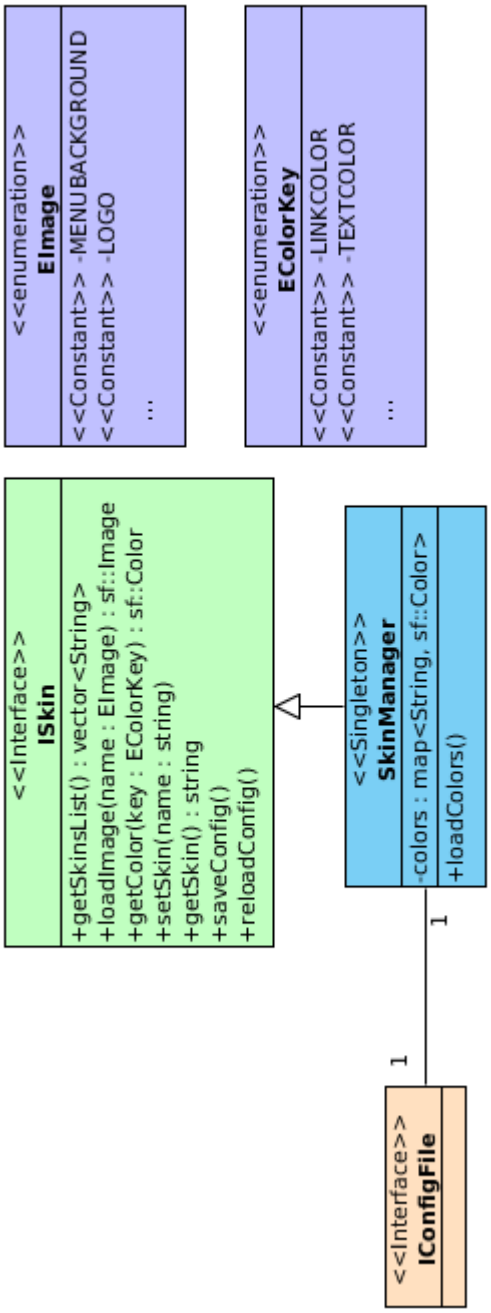
C.5 Module Moteur de jeu



C.6 Module Configuration



C.7 Module Skin



C.8 Module Son